Tero Elonen

Suorituskykyinen verkkosivu sisällönhallinta-järjestelmän avulla

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Mediatekniikan koulutusohjelma
Insinöörityö
20.5.2011

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Tero Elonen Suorituskykyinen verkkosivu sisällönhallintajärjestelmän avulla 60 sivua + 2 liitettä 3.6.2011
Tutkinto	insinööri (AMK)
Koulutusohjelma	mediatekniikka
Suuntautumisvaihtoehto	digitaalinen media
Ohjaaja	yliopettaja Harri Airaksinen tuotantopäällikkö Jesse Peurala

Insinöörityössä tutkittiin sisällönhallintajärjestelmällä rakennetun verkkosivun optimointia. Tavoitteena oli selvittää yleisesti jokaiselle sivustolle suoritettavat optimointitoimenpiteet sovellustasolla. Projekti tehtiin yhteistyössä verkkopalveluita toimittavan yrityksen kanssa ja optimointi sovellettiin yrityksen asiakasprojektiin.

Työssä selvitettiin yleisimmät tavat optimoida verkkosivuja ja etenkin sisällönhallintajärjestelmämillä rakennettuja verkkosovelluksia. Esimerkkinä oli Drupal-sisällönhallintajärjestelmä, jota käytettiin myös projektissa, joka optimoitiin.

Työn tuloksena saatiin selville yksinkertaiset ja tehokkaat tavat optimoida verkkosivua ja testata sen suorituskykyä. HTTP-pyyntöjen vähentämisen ja välimuistin oikean käytön tärkeys optimoinnissa tulivat hyvin esille. Drupal-sisällönhallintajärjestelmästä ja sen laajennusosista löydettiin useita suorituskykyä parantavia ominaisuuksia ja saatiin uutta tietoa Panels-moduulin vaikutuksesta sivuston suorituskykyyn. Teeman optimoinnista opittiin myös paljon jatkossa hyödyllistä tietoa, kuten HTTP-pyyntöjä vähentäviä tekniikoita.

Optimoitava projekti oli ekstranetverkkopalvelu, jonka käyttäjäkunta koostui ainoastaan sisään kirjautuneista käyttäjistä. Palvelu optimoitiin teeman, sovelluksen ja sen moduulien osalta. Palvelun toimintaa testattiin ennen optimointia ja sen jälkeen, jotta voitaisiin nähdä optimoinnin vaikutukset ja vetää niistä johtopäätöksiä tulevia optimointiprojekteja varten.

Insinöörityön lopputuloksena syntyi verkkopalvelu, joka kestää huomattavaa käyttäjäkuormaa ja jota on helppo kehittää tulevaisuudessakin. Projektin rajallisen aikataulun vuoksi kaikkia mahdollisia optimointitoimenpiteitä ei voitu palveluun tehdä, mutta sen toiminta saatiin optimoitua huomattavasti lähtökohtaa paremmaksi. Optimointi osoittautui erittäin kiinnostavaksi ja monivaiheiseksi projektiksi, joka tulisi huomioida heti projektin alussa. Sovellukseen tarkoituksenmukaisten moduulien valitseminen ja välimuistin tärkeys nousivat esille projektin aikana. Insinöörityön pohjalta projektin mahdollistanut yritys uudisti Drupal-sivustojensa optimointikäytäntöä ja ohjeistusta.

Avainsanat	optimointi, sisällönhallintajärjestelmä, välimuisti, Drupal

Metropolia

Abstract

Author Title Number of Pages Date	Tero Elonen Building high-performance web site on a content management system 60 pages + 2 appendices 3 June 2011
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructor	Harri Airaksinen, Principal Lecturer Jesse Peurala, Production Manager

This thesis is focused on optimizing a web site built on a content management system (CMS). The objective was to research common ways of optimizing on the software level. The project was conducted in co-operation with a software development company and the practical optimizing was applied on one of their projects.

The thesis describes the most common ways of optimizing web sites especially from the content management system point of view. The optimized web service was built on Drupal-content management system. Drupal was also the CMS that's optimizing was examined the most throughout the thesis.

As a result of the optimization project and the studies on the subject prior to the project itself, I was able to learn simple and effective ways to optimize a website and to test its performance. The importance of reducing HTTP-requests and the right use of cache became clear. I found new performance-boosting features from Drupal and its modules and I also uncovered new information about Panels-module and its impact on performance. I learned useful techniques about website theme-optimizing as well.

The project that was optimized was an extranet web service consisting entirely of logged in users. The theme, the software and its modules were all optimized. Before and after the optimization process, tests were run on the service to see the effects of the optimization and draw conclusions on their effectiveness for future projects.

As a result of this thesis, the web service can withstand higher user concurrency than before and further development of the project is easier. Due the tight schedule of the project, all possible optimizing procedures were not implied; however the performance of the service was greatly increased.

Optimizing turned out to be very interesting and multistage project that should be taken into consideration already in the beginning of the project. Choosing only appropriate modules and the importance of cache came clear during the process. Based on this thesis, the company that made this project possible renewed its optimizing procedure and guidelines.

Keywords	Optimizing, Content Management System, Cache, Drupal

Sisällys

1 Johdanto

Nykyihminen on hyvin kiireinen ja malttamaton varsinkin verkossa asioidessaan. Hitaasti toimivat verkkosivut ja palvelut verkossa turhauttavat kävijöitä siinä määrin, että pienikin lisäys sivun latautumisaikaan voi aiheuttaa suuren kävijäkadon [1]. Tähän verkkosivujen käyttäjien nopeusvaatimukseen ei ole mahdollista vastata, ellei verkkosivuja rakenneta nopeutta silmällä pitäen. Insinöörityön tavoitteena on tutkia verkkosivun optimointia sisällönhallintajärjestelmän näkökulmasta ja selvittää siihen liittyviä tekniikoita ja niistä saatavia etuja sivuston toiminnan nopeuttamiseksi. Lähes jokaista verkkosivun osaa voidaan optimoida nopeamman palvelun aikaansaamiseksi. Kun halutaan tarjota sivusto nopeasti käyttäjille, sen kaikkien osien tulee olla tasapainossa palvelimista lähtien. Tässä työssä keskitytään sovelluspuolen optimointiin eli sisällönhallintajärjestelmällä rakennetun verkkosivun toiminnan nopeuttamiseen. Palvelinpuolen optimointiin ei tässä työssä juurikaan paneuduta.

Insinöörityö tehdään Soprano Brain Alliance Oy:lle, joka on web-pohjaisten sovellusten ja järjestelmien toimittaja. Yritys on osa Soprano-konsernia, ja se työllistää 40 henkilöä. Yritys on erikoistunut PHP-ohjelmointikielellä toteutettujen järjestelmien toimittamiseen, ja sen palkkalistoilla on eniten PHP-sertifioituja ohjelmistokehittäjiä koko Suomessa. Järjestelmien kehittämisen lisäksi yritys tarjoaa resurssivuokrausta ja konsultointipalveluja niitä tarvitseville yrityksille. Vahvasta PHP-osaamisesta kumpuaa myös yrityksen tarjoama koulutus aiheesta ja siihen liittyvistä järjestelmistä. Yrityksen toiminta on hyvin teknologiapainotteista, ja siltä löytyy huippuosaamista monien verkkoteknologioiden osalta. Yrityksen asiakaskuntaan kuluvat suuret ja keski-suuret yritykset, jotka yleensä vaativat monimutkaisia ja laajoja järjestelmiä verkkoon. [2.]

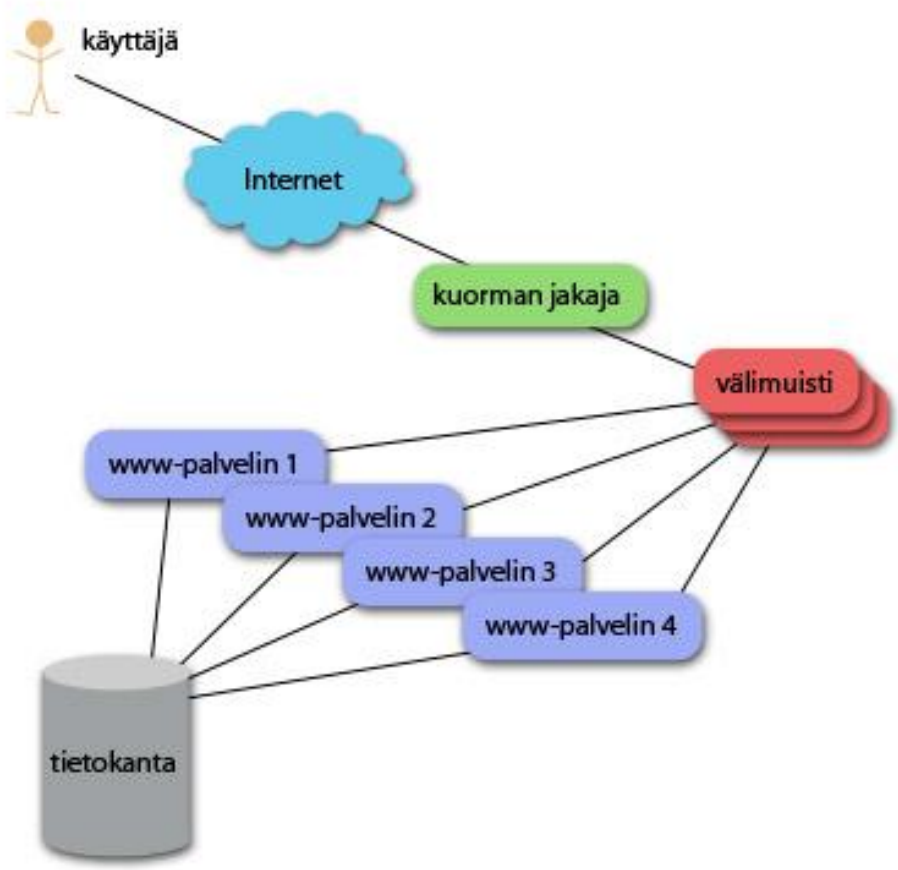
Insinöörityöraportissa käydään läpi verkkosovellusten sovelluspuolen optimointitekniikoita ja niiden suorituskyvyn mittaustekniikoita sekä tulosten analysointia. Työ käsittelee yleisesti verkkosivujen ja sisällönhallintajärjestelmien toimintaa sekä optimoinnin yleisiä pääperiaatteita ja tavoitteita. Raportissa tutkitaan tarkemmin sisällönhallintajärjestelmien optimointia ja sitä, miten verkkosovelluksen toimintaa voidaan nopeuttaa puuttumatta palvelimen toimintaan. Työssä tutkitaan optimoinnin tulosten mittaami-

seen tarkoitettuja työkaluja ja analysoidaan käytännön optimointityötäni sekä suorittamiani mittauksia Helsingin Lääkärikeskus Oy:lle tehdylle verkkopalvelulle.

2 Verkkosivut ja sisällönhallintajärjestelmät

2.1 Selaimen ja palvelimen välinen liikenne

Kun selataan verkkosivustoja, selain lähettää Internetin kautta pyyntöjä palvelimille, jotka tarjoavat verkkosivun osia pyyntöjä vastaan. Näitä pyyntöjä kutsutaan HTTP-pyynnöiksi (HyperText Transfer Protocol), ja ne ovat palvelimen ja selaimen välistä keskustelua. Suurimmassa osassa verkkosivuja selain ja verkkosivua tarjoava palvelin eivät kuitenkaan kommunikoi ilman välikäsiä, vaan liikenne kulkee kuorman jakajan lävitse, joka ohjaa liikenteen oikealle palvelimelle. [3, s. 106.] Kun selaimelta palvelimelle kulkeva liikenne on kulkenut kuorman jakajan lävitse, sitä voidaan ohjata takaisin selaimelle, ennen kuin se saavuttaa palvelimen. Tätä kutsutaan sivuston osien tarjoamiseksi välimuistista. [3, s. 109.] Kuvassa 1 on havainnollistettu yksinkertaisella mallilla, miten liikenne selaimen ja palvelimen välillä liikkuu.



Kuvio 1. Käyttäjän selaimen ja palvelimen välinen liikenne [3, s. 109].

Kun pyynnöt saavuttavat palvelimen, ne prosessoidaan ja niiden mukaiset tehtävät suoritetaan ennen vastauksen palauttamista selaimelle. Näihin tehtäviin lukeutuu usein tiedon hakeminen tietokannasta, joka voi sijaita samalla tai eri palvelimella, kuin mistä sivu tarjotaan.

2.2 Sisällönhallintajärjestelmät

Sisällönhallintajärjestelmä on termi, jota on vaikea yksiselitteisesti määritellä, mutta verkossa sillä yleisesti tarkoitetaan verkkosivun sisällön hallintaan kehitettyä sovellusta, joka parhaimmillaan parantaa verkkosivun käyttökokemusta niin kuluttajalle kuin kehittäjälle. Sen perimmäinen tarkoitus on julkaistun tiedon hallinta ja jakaminen halutulle yleisölle. [4.] Verkkosivujen hallintaan rakennettuja sisällönhallintajärjestelmiä kutsutaan usein myös julkaisujärjestelmiksi.

Kaikki kolme yleisintä sisällönhallintajärjestelmää, nimeltään WordPress, Joomla! ja Drupal, perustuvat vapaaseen lähdekoodiin, ja ne ovat ilmaiseksi ladattavissa verkosta [5; 6; 7; 8]. Näistä järjestelmistä monipuolisimpana ja muokattavimpana pidetään Drupalia. Drupal on sen ympärille rakentuneen yhteisön kehittämä ja ylläpitämä sovellus, josta viimeisin versionumero 7 julkaistiin tammikuussa 2011. Suurin osa Drupalin suosiosta perustuu varmasti sen laajennettavuuteen ja muokattavuuteen. Tämän ominaisuuden mahdollistavat yhteisön kehittämät ja ilmaiseksi saatavissa olevat järjestelmän laajennusosat. Nämä moduuleiksi kutsutut paketit parantavat Drupalin perusasennuksen mukana tulevia ominaisuuksia tai lisäävät järjestelmään uusia toiminnallisuuksia, joita siinä ei aiemmin ollut. [8.]

Drupal on rakennettu käyttäen PHP-ohjelmointikieltä, joka on laajasti käytetty varsinkin verkkosovelluksissa. PHP on ollut jo vuosia palvelinpuolella käytetyin ohjelmointikieli, ja kaikkiaan ohjelmointikielistä se on viidenneksi suosituin TIOBE-ohjelmointiyhteisön mukaan [9; 10]. Sitä käytettiin vuonna 2007 yli 20 miljoonassa verkkotunnuksessa Internetissä [11]. Kielen yleisyyden ansiosta monet kehittäjät ympäri maailmaa pystyvät kehittämään Drupalia ilman, että heidän tarvitsisi opetella uutta ohjelmointikieltä, ja tämä on myös yksi Drupalin vahvuuksista. Suuri määrä kehittäjiä mahdollistaa suuren

ominaisuuksien kirjon ja takaa paremman tuen julkistetuille moduuleille ja itse sisällönhallintajärjestelmälle. [8.]

Drupalin, kuten muidenkin suosittujen sisällönhallintajärjestelmien, ulkoasun voi muokata haluamansa näköiseksi luomalla sivustolle teeman. Teemoja ei ole kuitenkaan välttämätöntä tehdä itse, vaan niitä on saatavilla verkosta ilmaiseksi, ja perusasennuksenkin mukana tulee muutamia erilaisia teemoja. [8.]

2.3 Yleistä verkkosivun optimoinnista

Verkkosivun optimoinnilla tarkoitetaan verkkosivuston toiminnan nopeuttamista ja parantamista siten, että pystytään tarjoamaan nopeampi vaste käyttäjälle. Käyttäjien sietämästä odotusajasta on tehty useita tutkimuksia, ja niistä saadut tulokset poikkeavat hieman toisistaan. Esimerkiksi Fiona Nahin kirjoittama tutkimus vuodelta 2004 kertoo, että ilman minkäänlaista vastetta käyttäjän toimiin käyttäjät jaksoivat odottaa sivun latautumista keskimäärin 5–8 sekunnin ajan ensimmäisellä yrityksellä ja seuraavilla yrityksillä vain 2–3 sekunnin ajan. [12.]

Toinen optimoinniksi kutsuttu toimenpide on hakukoneoptimointi, jossa verkkosivusta pyritään tekemään hakukoneystävällinen, jotta sivusto saavuttaisi korkean sijan hakukoneiden hakutuloksissa. Hakukoneoptimointia ei tulisi kuitenkaan sekoittaa tavalliseen optimointiin, vaikka optimoitu sivusto usein on hakukoneystävällisempi kuin optimoimaton, sillä tavoitteet näissä optimoinnin tyypeissä ovat erilaiset. [13, s. 1, 5.]

Verkkosivun HTML-dokumentin (HyperText Markup Language) latausaika, jos ei lasketa tietojen prosessointia mukaan, on yleensä vain noin 10 % koko sivun latausajasta ja 80–90 % ajasta kuluu erilaisten sivun komponenttien, kuten kuvien, videoiden, flash-tiedostojen ja vastaavien, lataamiseen. Optimointi keskittyykin yleensä tähän 90 % sivunlatausajasta vievään osioon sivustolla. [14, s. 5.]

Optimointia voidaan tehdä sivustolle kahteen pääkohteeseen: palvelimelle ja sen ohjelmistolle tai itse verkkosovellukselle. Palvelimella on mahdollista lisätä laskentatehoa eli hankkia suorituskykyisempiä osia palvelimeen, kuten tehokkaampi prosessori ja lisää muistia, jotta prosessointi palvelimella nopeutuisi. Tämä nopeuttaa aikaa, jossa palvelin

pystyy palauttamaan vastauksen kyselyyn, ja mahdollistaa suurempien käyttäjämäärien vierailun sivustolla yhdellä kertaa. Myös palvelimella olevaa ohjelmistoa voidaan optimoida toimimaan nopeammin ja tarkoituksenmukaisemmin ja näin nopeuttaa koko sivuston toimintaa.

Itse verkkosovelluksen optimointi on pääasiassa sovelluksen muokkaamista sellaiseen muotoon, että se tekisi mahdollisimman vähän kyselyitä palvelimelta ja olisi nopeasti ladattavissa selaimelle, eli tiedostokooltaan pieni. Mitä enemmän pyyntöjä käyttäjän toimet sivustolla palvelimelle lähettävät, sitä enemmän liikennettä palvelimen ja selaimen välillä tapahtuu. Kommunikoinnin suuri määrä hidastaa verkkosivujen toimintaan, ja siksi näiden pyyntöjen määrää pyritään vähentämään. Tiedostokoon pienentäminen taas nopeuttaa yksittäisen komponentin lataamista verkosta, joten jos kaikki ladattavat komponentit saadaan mahdollisimman pieneen kokoon, säästetään aikaa. [14, s. 6–7, 10–11.] Verkkosivun koodin kirjoittaminen mahdollisimman tiiviisti ja tehokkaasti siis pienentää niiden aiheuttamaa latausrasitusta ja yleensä jopa parantaa koodin toimivuutta.

Sivuston ulkoiseen olemukseen liittyvät elementit, kuten kuvat ja videot, optimoidaan mahdollisimman pieneen kokoon, jotta niiden lataus veisi mahdollisimman vähän aikaa. Tämä on mahdollista pakkaamalla tiedostot jollakin häviöllisellä pakkausalgoritmilla siten, että tiedostokoko saadaan pienemmäksi hävittämällä elementeistä yksityiskohtia. Jos mahdollisuudet sallivat, osa tiedostoista voidaan jopa pakata tiedostonpakkausmenetelmällä kuten Gzip. Gzip on vapaan lähdekoodin pakkausapuohjelma, jolla voidaan pakata esimerkiksi sivuston tyylit määrittelevät tiedostot erittäin pieneen tilaan ja näin saada niiden tiedostokoko minimoitua. [15.]

Yksi tärkeimmistä optimoinnin kohteista on sisällön tarjoaminen käyttäjille välimuistista aina, kun mahdollista. Välimuistiin tallennettu sisältö voidaan hakea nopeasti käyttäjän selaimeen ilman, että sitä tarvitsee prosessoida samalla tavoin kuin palvelimelta tavanomaisesti haettava sisältö. Esimerkiksi jonkin tietokantahaun tulos voidaan tallentaa välimuistiin ja tarjota sieltä nopeasti käyttäjän selaimen sitä pyytäessä ilman, että itse tietokantahakua on suoritettava. Täytyy kuitenkin muistaa, että välimuisti toimii vasta siinä vaiheessa, kun samaa sisältöä haetaan toista kertaa, koska ensimmäisellä kerralla tietoa haettaessa aikaa vievät tietokantahaut ja prosessointi on suoritettava. Vasta tä-

män jälkeen se tallentuu välimuistiin, josta se voidaan tarjota nopeammin käyttäjälle. Sisällönhallintajärjestelmät käyttävät yleisesti tietokantaa sisällön varastoimiseen, joten on erityisen tärkeää huomioida välimuisti näitä järjestelmiä käytettäessä, tietokantakyselyn hitauden vuoksi. Varsinkin staattinen sisältö kannattaa tallentaa välimuistiin sen sijaan, että joka kerta kun sisältöä kysytään, tehtäisiin hidas tietokantahaku. [16, s. 349.]

Selaimissa on myös oma välimuisti, joka tallentaa kerran haettuja komponentteja sivulta. Verkkosovelluksen tehtävänä on kertoa selaimelle, mikä tieto on vanhentunutta ja mikä ei, jotta se voi tarjota oikean version käyttäjän nähtäväksi. Siksi on tärkeää säätää verkkosovelluksen asetukset siten, että voidaan hyödyntää selaimen välimuistia mahdollisimman tehokkaasti. Selain tarkistaa palvelimelta välimuistissa olevan tiedon oikeellisuuden, jos sen on epävarma komponentin vanhentumisajankohdasta. Tämän vuoksi on tärkeää käyttää komponenteissa elinajan määrittelyä. [14, s. 7–8.]

Content Delivery Network eli lyhennettynä CDN on yksinkertainen tapa parantaa sivuston toiminnan nopeutta käyttäjille ympäri maailmaa. CDN:n idea on tuoda sivusto käyttäjälle mahdollisimman läheltä. Sivusto tallennetaan ympäri maailmaa sijaitseville palvelimille niiden välimuistiin. Käyttäjän selatessa sivustoa katsotaan, missä päin käyttäjä sijaitsee, ja tarjotaan sivusto hänelle lähimmän mahdollisen palvelimen välimuistista. Suuria tällaisia palvelinverkostoja ovat Akamai ja Limelight Networks. Korkean hinnan vuoksi CDN:n käyttäjät kuitenkin rajautuvat yleensä vain suuriin yrityksiin kuten Yahoo!, joilla on varaa ostaa tällaisia palveluita. [13, s. 279.]

3 Sisällönhallintajärjestelmän optimointi

3.1 Optimoinnin pääperiaatteet

Sisällönhallintajärjestelmän optimoinnissa pätevät samat periaatteet kuin verkkosivun optimoinnissa muutenkin. Siinä voidaan usein käyttää optimointiin suunniteltuja valmiita osia, joilla sivuston suorituskykyä voidaan parantaa melko yksinkertaisilla toimenpiteillä. Valmiit osat ovat kuitenkin myös sisällönhallintajärjestelmien suurin kompastuskivi, joten ylimääräisten ja tarpeettomien osien asentaminen pahimmassa tapauksessa heikentää sivuston suorituskykyä. Tämän vuoksi ennen kuin sivustolle asennetaan suorituskykyä parantavia lisäosia, tulisi käydä kaikki sivuston lisäosat läpi. Niiden tarpeellisuus ja toiminta tulisi varmistaa, sillä hyödyttömien ja hitaiden lisäosien poistamisesta saattaa olla enemmän hyötyä kuin uuden suorituskykyä parantavan lisäosan asentamisesta [17].

Kun käytettään sisällönhallintajärjestelmää, kuten Drupalia, kasvaa HTTP-pyyntöjen määrä suureksi, varsinkin jos käytössä on useita moduuleja. Jokainen moduuli lisää yleensä yhden tai useamman HTTP-pyynnön palvelimelle esimerkiksi CSS-tiedoston muodossa. Optimaalista siis olisi, jos uusi, sivustolle lisättävä ominaisuus pystyttäisiin toteuttamaan jo olemassa olevilla moduuleilla. [18; 19.]

Julkisella sivustolla olevat kehitykseen käytettävät moduulit, kuten osoitteesta http://drupal.org/project/devel ladattavissa oleva devel-moduuli Drupalissa, tulisi vähintäänkin kytkeä pois tai mieluiten poistaa [20]. Näitä moduuleita ei ole tarkoitettukaan käytettäväksi julkisilla sivustoilla, vaan ne on puhtaasti suunniteltu sivuston kehitysvaihetta varten, joten niistä aiheutuva kuorma palvelimelle on suuri, jos ne jätetään julkiselle sivustolle. Monet moduulit myös lisäävät paljon graafisesti hienoja toiminnallisuuksia sivustolle, mutta ne tuovat usein mukanaan paljon JavaScriptiä, joka taas voi huonosti koodattuna hidastaa sivustoa [13, s. 217]. Kannattaa siis arvioida, onko moduulin tuoma graafinen loisto ja mahdollinen käytettävyyden parannus sen viemän suorituskyvyn arvoista.

Moduulien vähentämisen lisäksi hyvä tapa vähentää HTTP-pyyntöjä sivustolla on käyttää teeman kuvissa sprite-grafiikkaa ja CSS-tyyliohjeita. Niiden avulla teeman kuvatie-

dostot saadaan parhaassa tapauksessa ladattua yhdellä HTTP-pyynnöllä. Kuten kuvan 2 Googlen sprite-grafiikasta voidaan havaita, sprite-grafiikalla tarkoitetaan verkkosivujen tapauksessa kaikkien grafiikassa käytettyjen kuvien tallentamista ruudukkoon yhteen tiedostoon. Haluttu kuva poimitaan ruudukosta tyyliohjeiden avulla sivuston tiettyyn elementtiin. Näin vain tietty osa sprite-grafiikasta näkyy käyttäjälle. Kun kaikki kuvat ovat yhdessä tiedostossa, joudutaan palvelimelle tekemään vain yksi HTTP-pyyntö. [13, s. 160.] Kuvien lisäksi myös muita sivustolla olevia tiedostoja, kuten CSS-tyylitiedostoja, kannattaa pakata yhteen HTTP-pyyntöjen vähentämiseksi.

Kuvio 2. Googlen sprite-grafiikka vuodelta 2009 [21].

Yhteen pakkaamisen lisäksi kuvien tiedostokoko optimoidaan tallentamalla ne sopivalla tallennusformaatilla, joka käyttää häviöllistä kuvanpakkausalgoritmia. Riippuen häviöllisyyden määrästä kuvien kokoa voidaan pienentää alkuperäisestä huomattavasti. Yleinen tallennusformaatti on JPEG, joka on lyhenne sanoista Joint Photographic Experts Group. Se käyttää häviöllistä pakkausalgoritmia, joka hävittää kuvasta näyttämisen kannalta tarpeettomia yksityiskohtia. Mitä suurempi pakkaustaso kuvassa on, sitä pienempi tiedostokoko saavutetaan, mutta myös kuvanlaatu huononee. [22.] Tallennettaessa kuvia verkkosivulle kannattaa käyttää kuvankäsittelyohjelmaa, kuten Adobe Photoshop, kuvien tallentamiseen ja tallentaa ne mahdollisimman suurella pakkaustasolla, mutta kuitenkin niin, että kuvat näyttävät hyvältä. Kuvasta riippuen tallennusformaattia vaihtamalla voidaan saada hyvännäköinen kuvanlaatu pienemmällä tiedostokoolla, joten optimaalisen koon saavuttamiseksi tulisi valita sopiva kuvaformaatti. [13, s. 167–169.]

Musiikkia ja videoita tarjoavilla verkkosivuilla näiden mediatiedostojen optimointi on tärkeää. Tiedostoko voi videotiedostoissa kasvaa helposti suureksi, joten sopivan videokuvan koon ja pakkaustason käyttäminen on tärkeää siedettävien latausaikojen

saavuttamiseksi. [13, s. 169–170.] Oman www-palvelimen kuorman vähentämiseksi optimoitaessa kannattaa harkita videoiden tarjoamista jonkin siihen erikoistuneen palvelun, kuten YouTube, kautta. Tällöin palvelu pakkaa ja tarjoaa videot käyttäjille, eikä oman palvelimen tarvitse täyttää videoiden tarjoamiseen vaadittavia teho- ja tilavaatimuksia. [23.] Musiikkiin pätevät samat säännöt kuin muunkin multimedian pakkaamiseen, eli mitä häviöllisempi pakkaus, sitä pienempi tiedostokoko. Tämän takia verkkosivuilta ladattavaa musiikkia tai videoita optimoitaessa ne kannattaa pakata useammalla eri pakkaustasolla ja antaa käyttäjän valita haluamansa äänenlaatu [13, s. 170].

Tärkeä osa sisällönhallintajärjestelmällä rakennetun verkkosivun optimointia on välimuistin käyttäminen kaiken sisällön tarjoamiseen, jossa se on järkevää. Sisällönhallintajärjestelmissä, kuten Drupalissa, on usein sisäänrakennettu välimuisti ja sen toiminta on melko automaattista, mutta se täytyy muistaa ottaa käyttöön julkisilla sivustoilla. Sisäänrakennetun välimuistin lisäksi on mahdollista käyttää myös muilla kuin sovellustasolla sijaitsevia välimuistitekniikoita, kuten käänteistä välityspalvelinta. Näiden välimuistitekniikoiden yhteiskäytöllä voidaan saada aikaan nopeasti toimiva ja monille käyttäjille monistettavissa oleva järjestelmä, jota voidaan muokata sovelluksen tarpeisiin sopivaksi. Drupaliin on myös tarjolla moduuleita, joilla voidaan parantaa sen välimuistitoimintoja ja lisätä sivuston nopeutta varsinkin anonyymeille käyttäjille. [16, s. 349–353; 3, s. 105–110.]

3.2 Teeman tyylien ja dynaamisuuden optimointi

Verkkosivun ulkoasu ja sen sisältämät visuaaliset elementit, joita ei ole rakennettu nopeutta mielessä pitäen, ovat usein osasyyllisiä sivuston hitaaseen toimintaan. Niiden optimointi on hyvä aloittaa heti sivuston rakennusvaiheessa, koska jälkeenpäin monimutkaisen teeman optimointi on todella työlästä. Sama pätee myös sivuston dynaamisuuden mahdollistaviin osiin varsinkin, jos sivulle tulee paljon tällaisia toiminnallisuuksia.

CSS-tyyliohjeet

CSS- (Cascading Style Sheets) eli kaskadisia tyyliohjeita käytetään verkkosivustoilla tyylien, kuten typografian, kuvien asettelun ja värimaailman, määrittelemiseen [24].

Selkeä CSS-arkkitehtuuri helpottaa sivujen ylläpitoa varsinkin, jos sivuston ulkoasuun tehdään muutoksia. Usein nämä tyylimääritykset tehdään erilliseen tiedostoon, mutta määrityksiä voidaan kirjoittaa suoraan myös HTML-koodin sekaan. Erillisten tyylitiedostojen avulla sivuston rakenteesta saadaan kuitenkin selkeämpi ja hakukoneystävällisempi, kun HTML-koodin sekaan kirjoitetut tyylit eivät ole sekoittamassa itse HTML-koodin luettavuutta. Erillisiin tiedostoihin kirjoitetut tyylitiedostot tulisi ladata sivuston ylätunnisteessa, jotta sivuston latautuessa elementit näyttäisivät suoraan tyylitiedostossa määritellyn mukaisilta [13, s. 186, 189].

Tyylitiedostoja voi optimoida monella tapaa, mutta onnistunut optimointi vaatii tyylimääritysten logiikan ja toiminnan ymmärrystä. Kuten esimerkkikoodista 1 käy ilmi, CSS-syntaksi koostuu kolmesta pääosasta: valitsimesta, ominaisuudesta ja ominaisuudelle annetusta arvosta. Valitsimella valitaan haluttu HTML-elementti, johon ominaisuus halutaan kohdistaa. Ominaisuus ja sille määritetty arvo erotetaan toisistaan kaksoispisteellä, ja ominaisuuden arvon määrittely päätetään puolipisteellä. Yhdellä valitsimella voi olla yksi tai useampia ominaisuuksia. [25; 13, s. 187.]

```
valitsin{ominaisuus:arvo;}
```

Esimerkkikoodi 1. CSS-syntaksi.

Valitsimena voidaan käyttää elementin HTML-tagia, elementille määritettyä id-arvoa tai luokka-arvoa. Näiden perusvalitsinten lisäksi on monia edistyksellisiä tapoja valita haluttuja elementtejä, kuten esimerkiksi tietyn elementin lapsi-elementti. Valitsimeen voidaan myös määrittää useampia arvoja, jolloin pystytään valitsemaan useampia elementtejä kerralla. Tällöin elementit, joihin kohdistuu samat ominaisuudet, voidaan kirjoittaa lyhyesti pilkulla erotettuina. Esimerkkikoodista 2 käy ilmi, miten valitsimia ketjuttamalla koodista saadaan lyhyempää.

```
h1,h2{color:#000;}
```

Esimerkkikoodi 2. Valitsimien yhdistäminen.

Valitsimen lisäksi myös ominaisuuksille annettuja arvoja voidaan kirjoittaa monella tapaa, mutta kannattaa valita lyhin, jolloin tyylitiedoston koko ei turhaan kasva. Kuten esimerkkikoodista 3 nähdään, on kolme eri tapaa kirjoittaa sama ominaisuus tietylle elementille, joista viimeinen on optimoinnin kannalta paras lyhyytensä ansiosta.

```
.esimerkkivalitsin {
margin-left:2em;
margin-right:2em;
margin-top:2em;
margin-bottom:2em;}

.esimerkkivalitsin2 {margin:2em 2em 2em 2em;}

.esimerkkivalitsin3 {margin:2em;}
```

Esimerkkikoodi 3. Eri tapoja kirjoittaa sama ominaisuus tyylitiedostoon.

CSS-tiedostojen optimoinnissa pätee sama sääntö kuin muussakin optimoinnissa, eli asiat tulisi pitää mahdollisimman yksinkertaisina ja lyhyinä. Mitä enemmän määrityksiä tiedosto sisältää, sitä suuremmaksi sen tiedostokoko kasvaa. Tämän takia tulisi käyttää mahdollisimman paljon hyödyksi CSS-syntaksin tarjoamia keinoja lyhentää määrityksiä ja määritellä samanlaiset tyylit sisältävät elementit yhdessä. Valitsimena käytetyt pitkät ID-arvot ja luokka-arvot tulisi mahdollisuuksien mukaan lyhentää ja tarpeettomat kommentit CSS-tiedostossa poistaa. [13, s. 189.]

HTML-koodin sekaan kirjoitettuja CSS-määrityksiä tulisi välttää jo senkin takia, että niitä on hankala jälkikäteen muokata, mutta erityisesti jos samat määreet toistuvat koodissa useasti. Tämä kasvattaa turhaan HTML-tiedostojen kokoa ja hankaloittaa koodin luettavuutta. Samanlaista tyyliä käyttävien elementtien CSS-määritykset voidaan optimoida määrittämällä näille elementeille sama luokka, jonka tyylit määritellään sitten erillisessä CSS-tiedostossa kertaalleen ja kaikki tämän luokan elementit saavat tiedostossa määritellyt tyylit. [13, s. 189–190.]

Sisällönhallintajärjestelmissä käytetään teemoja ulkoasun muokkaamiseen, ja siksi suurin osa CSS-tiedostojen optimoinnista tehdään käytettävässä teemassa. Drupalissa nämä teemat sisältävät tyylitiedostot, suurimman osan ulkoasun kuvista, JavaScript-tiedostot teemaa varten ja teeman omia PHP-tiedostoja ja infotiedostoja. Sivuston ulkoasun rakentamiseen voi valita useita eri teemapohjia tai rakentaa oman teemansa itse. Valmiin teeman kopiointi ja muokkaaminen oman näköiseksi teemaksi on myös yksi mahdollisuus. [12, s. 74–93.] Optimoinnin kannalta suositeltavin lähestymistapa on käyttää valmista perusteemaa, kuten esimerkiksi Fusion-teema, joka on ladattavissa osoitteesta http://drupal.org/project/fusion. Fusion-teemassa mukana tuleva CSS-tiedosto sisältää valmiiksi muutamia perusmäärityksiä, jotka nollaavat selainten mukana tulevia oletustyylejä. Näin selainten eroavista oletusasetuksista päästään eroon ja voidaan aloittaa tyylittely samasta lähtöpisteestä. Tyhjän perusteeman käyttäminen on myös optimoinnin kannalta lähes välttämätöntä, koska kopioitaessa jokin jo olemassa oleva, valmis teema, ovat sen CSS-tiedostot harvoin optimaalisia ja niiden muokkaaminen uuden ulkoasun mukaisiksi jättää sivustolle helposti turhia tyylimäärityksiä vanhasta teemasta. [26.]

Teemat eivät ole ainoita CSS-tiedostoja sisältäviä Drupalin osia. Myös monet Drupalin ytimeen sisällytetyt ja Internetistä jälkikäteen ladatut moduulit lisäävät järjestelmään omat CSS-tiedostonsa. Nämä tiedostot kannattaa myös huomioida optimoitaessa sivustoa ja poistaa ne sivuston ylätunnisteesta, jos ne ovat tarpeettomia tai niiden sisältämät määritykset on jo määritelty muualla, kuten teeman CSS-tiedostoissa. [27.]

Drupal-sisällönhallintajärjestelmän suorituskykyvalikosta löytyy mahdollisuus parantaa sivuston nopeutta vähentämällä CSS-tiedostojen määrää yhdistämällä ne. Kun tämä ominaisuus on kytketty päälle, Drupal pakkaa siihen asennettujen moduulien CSS-tiedostot poistamalla välilyöntejä, rivinvaihtoja ja kommentit. Tämän jälkeen Drupal yhdistää kaikki CSS-tiedostot yhdeksi tiedostoksi ja käyttää tätä tiedostoa tyylien määrittelyyn sivustolla. Tämä ominaisuus vähentää HTTP-pyyntöjen määrää ja nopeuttaa sivustoa varsinkin, jos käytössä on paljon moduuleita, jotka käyttävät omia CSS-tiedostoja tai sivuston teemassa on paljon CSS-tiedostoja ja -määrityksiä. [28; 16, s. 536.]

Drupalin ytimen mukana tulevan CSS- ja JavaScript-optimoinnin parantamiseksi on tarjolla moduuli, jonka avulla yhteen kootut CSS- ja JavaScript-tiedostot voidaan pakata Gzip-muotoon. Kun tiedostot pakataan, moduuli tekee yhdistetyistä CSS- ja JavaScript-tiedostoista entistä pienempiä ja näin ollen nopeuttaa sivuston toimintaa. [32; 33.]

JavaScript-komentosarjakieli

JavaScriptin optimoinnissa on paljon samaa kuin CSS-tiedostojen optimoinnissa. Kommentit, turhat välilyönnit ja rivinvaihdot tulisi poistaa julkisilta sivustoilta JavaScript-tiedostokoon pienentämiseksi. Rivinvaihtoja karsittaessa tulisi kuitenkin muistaa, että JavaScriptissä rivinvaihto voidaan tulkita samaan tapaan kuin puolipiste, joka päättää koodirivin. JavaScript on muiltakin osin täynnä valinnaista syntaksia, joka selkeyttää koodin luettavuutta, mutta ei ole pakollista koodin toimivuuden kannalta. Esimerkiksi yksinkertaisessa ehtolauseessa, jossa on vain yksi toteamus, voidaan aaltosulut jättää kirjoittamatta. [13, s. 233.]

JavaScriptistä löytyy useita lyhenteitä joilla koodin määrää voidaan vähentää ja näin pienentää tiedostokokoa. Yksinkertainen ehtolause esimerkkikoodissa 4 voidaan kirjoittaa ilman aaltosulkuja. Ehtolause voidaan kirjoittaa myös hyvin lyhyesti yhdellä rivillä. [13, s. 231–232.]

```
var num;
if (x > 1)
{
  num = true;
}
```

Esimerkkikoodi 4. Yksinkertainen ehtolause, jossa voidaan jättää aaltosulut kirjoittamatta.

Muita tapoja vähentää JavaScript-tiedostojen kokoa on lyhentää pitkät muuttujien ja funktioiden nimet. Tämä kannattaa tehdä kuitenkin koodiin vasta kehitysvaiheen jälkeen, koska yhden kirjaimen muuttujan nimet ovat harvoin kuvaavia, joten ne haittaavat jatkokehitystä. [13, s. 233–234.] JavaScriptin pakkaamiseen löytyy myös työkaluja, kuten w3complier, joka tekee koneellisesti koodissa olevien muuttujien lyhentämisen

[29]. Näin kehittäjät voivat käyttää koodissaan kuvaavia muuttujan nimiä ja vasta ennen julkaisua suorittaa ohjelman, joka pakkaa koodista mahdollisimman tiiviin paketin. Esimerkkikoodissa 5 kirjoitettu ehtolause on pystytty lyhentämään yhden rivin mittaiseksi poistamalla ylimääräiset sulut ja käyttämällä lyhenteitä [13, s. 231–232].

```
var num;
if (x > 1)
{
  num = true;
}
else
{
 num = false;
}

var num = (x > 1) ? true : false;
```

Esimerkkikoodi 5. Ehtolause kirjoitettuna tavallisesti, ilman lyhenteitä ja lyhennettynä.

Koodia kirjoitettaessa syntyy usein myös tiettyjä koodinpätkiä, jotka toistuvat moneen kertaan. Tällaisista useasti esiintyvistä merkkijonoista voidaan tehdä makro eli tietystä komennosta syntyvä merkkijono, jota voidaan toistaa niin monesti, kuin on tarve. Koodinpätkän tulisi kuitenkin esiintyä tekstissä riittävän useasti, että makrosta saadaan vastaavaa hyötyä tiedostokokoa ajatellen. [13, s. 232.] Lyhenteiden lisäksi JavaScriptiä kirjoitettaessa voidaan olettaa tiettyjä oletusarvoja esimerkiksi funktiossa, joka hakee palvelimelta vahvistusta GET-metodia käyttäen. GET-metodi on tapa lähettää ja vastaanottaa tietoa palvelimelta osoiteriviin lisättävien argumenttien avulla [30]. Sen palautuksen oletusarvo on true eli tosi, joten tämän voi olettaa funktiota kirjoitettaessa ja jättää sen kirjoittamatta funktion määritykseen. [13, s. 236.]

JavaScriptin yleisesti aiheuttama ongelma on muistivuodot varsinkin vanhemmissa selainversioissa. Tämän välttämiseksi käytettyjen objektien arvot tulisi asettaa tyhjiksi ja käyttämättömät objektit poistaa, jotta JavaScriptin käyttämä järjestelmä pystyy vapauttamaan näiden objektien viemän muistin uudelleen käyttöön. [13, s. 241.]

Toinen JavaScriptiin liitetty uskomus on sen toimimattomuus välimuistin kanssa. Tämä on jossain määrin totta varsinkin selainten kohdalla, koska JavaScriptin ja sen pyyntöjen välimuistiin tallentaminen vaihtelee selaimesta riippuen. Tämän vuoksi JavaScript pyritään usein pitämään poissa selainten välimuistista, ja tähän ongelmaan on kehitetty useita ratkaisuja, kuten pyyntöjen polun muuttaminen uniikiksi joka pyynnön kohdalla. Tällöin selain ei voi käyttää jo välimuistissa olevaa vastausta pyyntöön, koska sen polku on eri kuin uudessa pyynnössä, jolloin pyyntö lähetetään jälleen palvelimelle. Ilman välimuistia optimoiminen ei kuitenkaan ole tehokasta, joten JavaScriptiä optimoitaessa oman välimuistin rakentaminen on yksi ratkaisu ongelmaan. Halutut pyynnöt ja niiden vastaukset voidaan tallentaa koodissa yksinkertaiseen taulukkoon. Koodin suorittaessa pyynnön, johon tavallisesti vaadittaisiin pyyntöä palvelimelle, tarkastetaan ensin välimuistiksi luotu taulukko. Jos pyyntöön löytyy vastaus jo taulukosta, erillistä pyyntöä palvelimelle ei tarvita. [13, s. 248–249.]

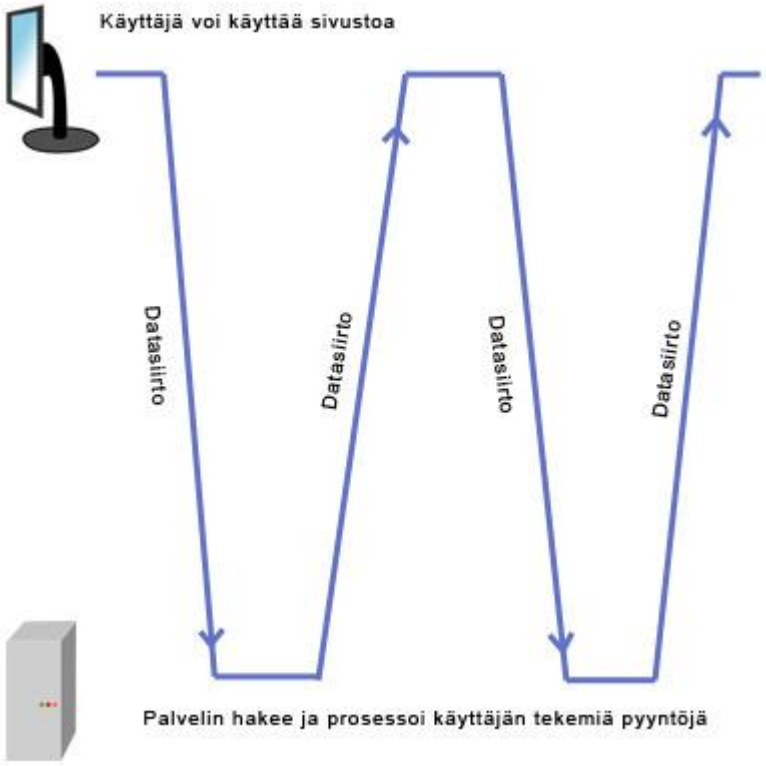
Optimoitaessa JavaScriptiä kannattaa huomioida, että kirjoitetun koodin tulisi olla nopeaa suorittaa ja myös kehittää. JavaScript-kirjastot, kuten jQuery ja Prototype, tarjoavat mahdollisuuden nopeaan kehittämiseen hyvin pienellä vaikutuksella suorituskykyyn. JavaScript-kirjastolla tarkoitetaan yhteen koottua JavaScript-koodia, joka helpottaa JavaScript-sovellusten tekoa ja tarjoaa työkaluja kirjoittaa koodi lyhyesti [31]. Kirjastojen lisääminen luonnollisesti lisää yhden tai useamman HTTP-pyynnön sivustolle, mutta niistä saatu hyöty on usein sen arvoista. Varsinkin jos JavaScriptiä tulee sivustolle paljon, kirjaston käyttäminen lyhentää koodia huomattavasti, jolloin jatkokehitys on pienemmän koodimäärän ansiosta helpompaa. JavaScript-kirjastot sisältävät usein suuren määrän koodia, mutta onneksi suurin osa kirjastoista on myös tarjolla pakatussa muodossa, jolloin niiden koko ei hidasta sivun latautumista suuresti. [13, s. 226–230.] Drupalin ytimeen on asennettu jQuery-kirjasto, joka aktivoituu automaattisesti jos sivustolle lisätään JavaScriptiä [16, s. 381]. Tämän takia varsinkin Drupalia käytettäessä kannattaa tutustua jQuery-kirjastoon ja optimoida JavaScript käyttämään jQuery-kirjaston tarjoamia lyhenteitä ja työkaluja.

Ajax-ohjelmointitekniikka

Nykyaikaisilla verkkosivuilla dynaamisuuden ja sulavan käytön mahdollistamiseksi käytetään usein teknologiaa nimeltä Ajax. Ajax on lyhenne sanoista *Asynchronous JavaSc-*

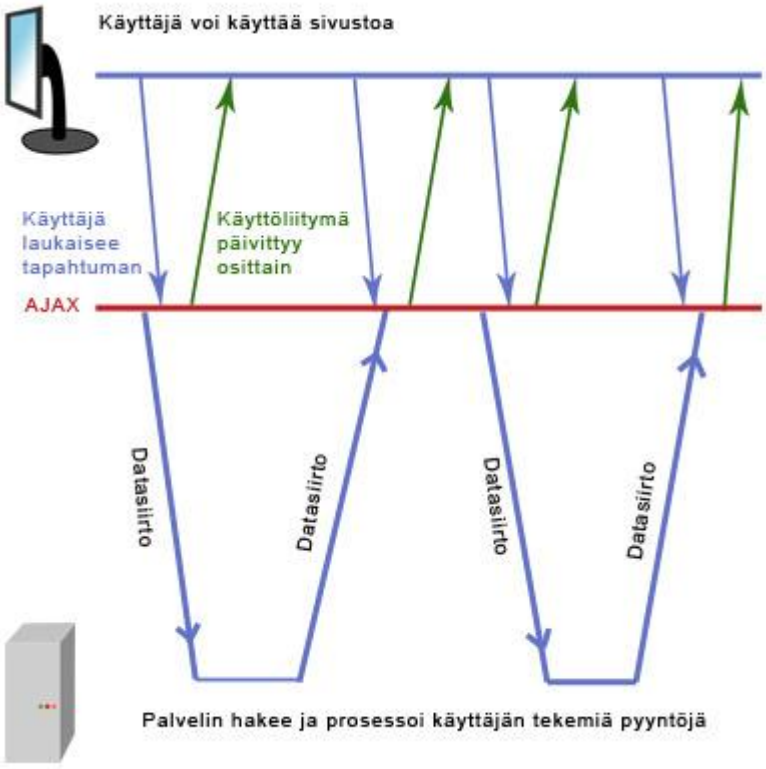
ript And XML, ja sillä tarkoitetaan käyttäjän selaimen ja palvelimen välillä toimivaa tasoa. Tämä taso mahdollistaa tiedon liikkumisen selaimen ja palvelimen välillä ilman sivunlatausta, eli osia sivusta voidaan päivittää lataamatta kaikkea uudestaan. Tämä vaikuttaa usein sivuston käyttökokemukseen positiivisesti, koska käytöstä tulee sulavampaa eikä turhaa odottelua esiinny niin usein. [13, s. 216.]

Kuviosta 3 nähdään, että käyttäjä joutuu odottelemaan palvelimelle lähetettyyn kyselyyn vastausta voimatta käyttää sivustoa samalla. Joka kerta, kun käyttäjä tekee muutoksen sivustolla, joka lähettää pyynnön palvelimelle, tapahtuu kuvassa näkyvä datasiirto. Tämän seurauksena sivu, jolla käyttäjä on, ladataan uudestaan. Näin ollen esimerkiksi käyttäjän täyttäessä lomaketta täytyy lomake lähettää kokonaisuudessaan palvelimelle tarkistettavaksi sen sisältämän tiedon oikeellisuuden toteamiseksi. Tämän jälkeen palvelin palauttaa uuden sivun käyttäjälle, joka kertoo mahdollisista puutteista lomakkeessa, minkä jälkeen käyttäjän on mahdollista korjata nämä virheet ja lähettää lomake uudestaan.



Kuvio 3. Sivuston toiminta ilman Ajax-teknologiaa [13, s. 219].

Kuviossa 4 on sivun toimintaperiaate, kun käytetään Ajaxia välittämään tietoa käyttäjän ja palvelimen välillä. Sivu toimii huomattavasti dynaamisemmin kuin ilman Ajaxia, ja käyttäjä pysyy jatkuvasti samalla sivulla. Sivu ladataan kokonaisuudessaan käyttäjän saapuessa sivulle, ja se antaa käyttäjälle palautetta muutoksista ilman, että kaikki tieto olisi lähetettävä palvelimelle palautteen saamiseksi. Ajax siis hakee tietoa palvelimelta piilossa käyttäjältä ja palauttaa tiedon dynaamisesti muuttamalla käyttöliittymää tarvittavilta osin. [13, s. 218.] Hyvänä esimerkkinä on lomakkeen täyttäminen, jossa pyydetään käyttäjää antamaan sosiaaliturvatunnuksensa. Käyttäjän kirjoittaessa sosiaaliturvatunnustaan sille osoitettuun kenttää käynnistyy Ajax-tapahtuma, joka lähettää kirjoitetun merkkijonon palvelimelle, joka taas prosessoi sen eli esimerkiksi tarkistaa sen oikeellisuuden ja palauttaa tuloksensa. Ajax päivittää käyttöliittymää kirjoittamalla esimerkiksi kentän viereen tekstin ”Sosiaaliturvatunnuksessa on virhe”. Käyttäjä voi nyt korjata kirjoitusvirheensä ilman sivunlatausta heti kentän täyttämisen jälkeen.



Kuvio 4. Ajax-teknologiaa käyttävän sivun toiminta [13, s. 219].

Ajaxin palvelimen ja selaimen välinen kommunikaatio perustuu pääasiassa *XMLHttpRequest*-pyyntöön. XMLHttpRequest lyhennetään yleisesti muotoon XHR, ja sen avulla on mahdollista lähettää palvelimelle pyyntö, johon palvelin luonnollisesti ideaalitapauksessa vastaa. [13, s. 222.] Tämä vastaus on usein XML-muotoista dataa, josta on johdettu x-kirjain Ajax-sanassa. XML-data on usein kuitenkin melko tilaa vievää, koska se koostuu XML-rakenteesta, jota ei välttämättä tarvita sovelluksessa. Sen lisäksi että XML-rakenne luo enemmän bittejä tiedostokokoon, se myös on parsittava ennen sen sisältämän tiedon hyödyntämistä. [13, s. 245.] Tähän XML:n ongelmaan vastaa JSON (lyhenne sanoista JavaScript Object Notation), joka on XML-formaattia yksinkertaisempi tapa välittää tietoa [34].

Kuten koodiesimerkkejä 6 ja 7 vertailtaessa nähdään, JSON-muotoinen taulukko on huomattavasti XML-muotoista taulukkoa yksinkertaisempi rakenteeltaan. JSON on myös nopea ja yksinkertainen parsia, joten optimoitaessa Ajaxia sivustolla, XML-dataa tulisi käyttää vain, jos siihen on hyvä syy, mutta muissa tapauksissa kannattaa valita kevyempi ja yksinkertaisempi JSON, koska se on optimaalisempi vaihtoehto.

```
[”value1”,”value2”,”value3”,”value4”]
```

Esimerkkikoodi 6. JSON-muotoinen taulukko.

```
<?xml version=”1.0” encoding=”UTF-8” ?>
<packet>
  <item>value1</item>
  <item>value2</item>
  <item>value3</item>
  <item>value4</item>
</packet>
```

Esimerkkikoodi 7. XML-muotoinen taulukko.

Ajaxilla on mahdollista tehdä myös sovelluksia, jotka näyttävät päivittyvän samanaikaisesti, kun palvelin on ottanut vastaan muuttuneen tiedon. Tällaisia sovelluksia kutsutaan myös push-sovelluksiksi, eli tieto työnnetään palvelimelta sovellukseen välittömäs-

tä sen saavuttua palvelimelle. Ajaxilla tämä kuitenkin täytyy toteuttaa lähettämällä palvelimelle kysely, joka kertoo, ovatko tiedot palvelimella muuttuneet. Jatkuva, tiheä kyselyiden teko palvelimelta voi aiheuttaa ongelmia varsinkin, jos moni käyttäjä käyttää sovellusta yhtäaikaisesti. Tämän takia tällä tekniikalla toimivat sovellukset kannattaa optimoida pienentämään kyselyiden määrää siinä tapauksessa, jos palvelimen vastaus on hidas. Vastausten nopeutuessa kyselyiden määrää voidaan jälleen nostaa, koska voidaan olettaa, että palvelimen kuorma on vähentynyt. On myös hyvä tapa informoida käyttäjää siinä tapauksessa, että Ajax-pyyntöön vastaaminen kestää, sillä usein käyttäjät odottavat välitöntä palautetta sovellukselta. [13, s. 253–254.]

Hyvistä puolistaan huolimatta Ajax on vaarallinen työkalu väärin käytettynä. Se voi hidastaa sivun latautumista, vaikeuttaa hakurobottien toimintaa ja aiheuttaa hidastelua, vaikka käyttäjä odottaisikin saavansa sivustolta välittömän palautteen toimilleen. Ajaxia käytettäessä tulisikin siis miettiä tarkkaan, onko sen käyttö tarkoituksenmukaista. [13, s. 217.]

3.3 Välimuisti

Tiedostokoon pienentämisen ja erillisten tiedostojen yhdistämisen lisäksi julkisella sivustolla on erityisen tärkeää muistaa kytkeä päälle sivuston välimuisti. Tällöin kaikki liikenne ei kulje suoraan www-palvelimelle suoritettavaksi, vaan osa pyynnöistä pystytään ohjaamaan takaisin selaimelle jo niiden saavuttua välimuistikerrokselle [35]. Välimuistia on monenlaista, ja sitä käytetään kaikkialla tietotekniikassa, kuten prosessoreissa ja kiintolevyissä. Onkin siis varsin luonnollista, että myös verkkosovellus sisältää välimuistia monessa eri järjestelmän tasossa. [3, s. 106.] Sovelluksessa sijaitsevan välimuistin lisäksi esimerkiksi MySQL-tietokannan sisältämä välimuisti on oma tasonsa, mikä nopeuttaa tietokantakyselyiden toimintaa. [16, s. 349.]

Drupalin tapauksessa järjestelmään on sisäänrakennettu välimuisti, joka toimii anonyymeille käyttäjille, mutta sisään kirjautuneille käyttäjille tieto ei tule järjestelmän välimuistista, vaikka se olisi kytketty päälle. Tämä johtuu siitä, että sisään kirjautuneilla käyttäjillä on usein yksilöllinen sisältö toisin kuin anonyymeillä käyttäjillä, jolloin tiedon tallentaminen välimuistiin olisi paljon tehottomampaa, koska kaikki tieto olisi tallennettava erikseen jokaiselle käyttäjälle. [16, s. 349.]

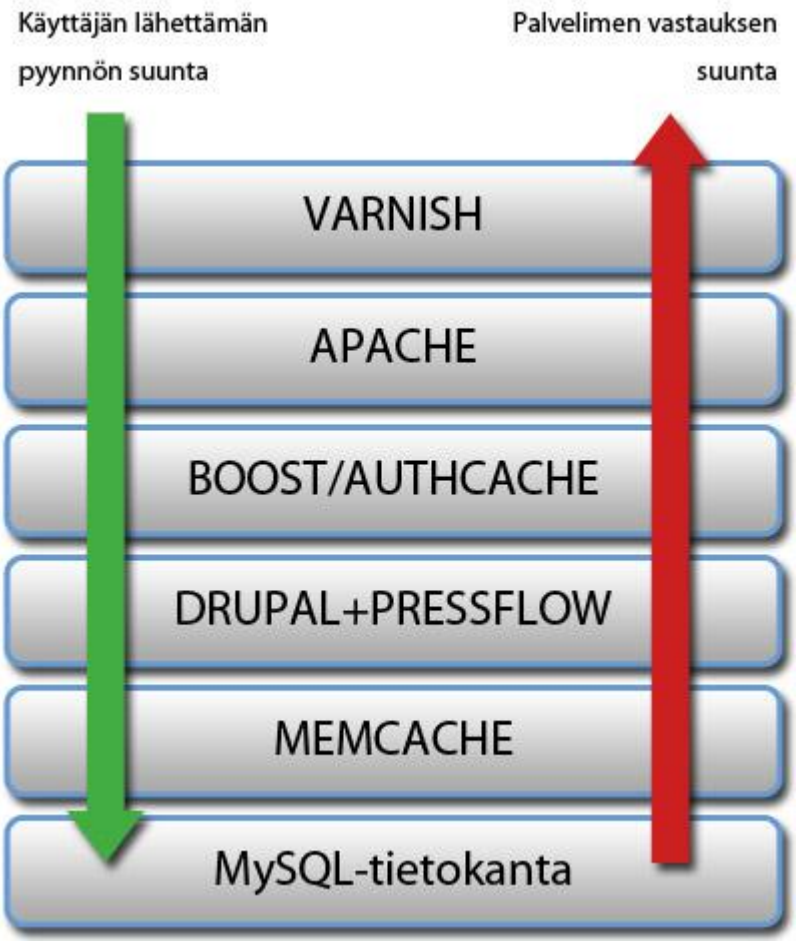
Drupalin sisäänrakennettu välimuisti käyttää tietokantaa tiedon varastoimiseen. Järjestelmä tallentaa palvelimelle saapuneiden kyselyiden vastauksia tietokantatauluun, josta ne voidaan tarjota samanlaisen kyselyn tullessa uudestaan vastaa, ilman että niiden saamiseen tarvittavia raskaampia kyselyitä ja laskutoimituksia tarvitsee suorittaa. Drupalin sisäänrakennettu välimuisti pystyy siis tarjoamaan tiedon yhdellä yksinkertaisella tietokantapyynnöllä tiedon, jonka hakemiseen tarvittaisiin muussa tapauksessa parhaimmillaan useita tietokantakyselyitä ja koodin suorittamista palvelimella. [16, s. 350.]

Osa Drupalin sisäänrakennetusta välimuistista toimii automaattisesti, eikä sen toimintaan voi vaikuttaa suorituskykyvalikosta. Tämä osa tallentaa välimuistiin Drupalin sisäisten komponenttien tietoja, kuten suodattimien, valikoiden ja moduulien asetuksia. Nämä tiedot tallennetaan aina välimuistiin, vaikka sivuston varsinainen välimuisti olisi kytketty pois päältä suorituskykyvalikosta. Järjestelmän automaattisesti erillisiin välimuistitauluihin tallentamat tiedot voidaan kuitenkin poistaa tyhjentämällä sivuston välimuisti ja käskemällä Drupal rakentamaan valikot uudestaan. Tällöin järjestelmä poistaa vanhan tiedon välimuistitauluista ja tallentaa uuden, ajantasaisen tiedon tilalle. [16, s. 351.]

Drupalin tarjoama sisäänrakennettu välimuisti ei aina riitä varsinkaan suurille sivustoille, joissa on myös paljon sisään kirjautuneita käyttäjiä. Drupalin sisäänrakennettu välimuisti on myös riippuvainen palvelimella sijaitsevista kiintolevyistä ja niiden kirjoitus- ja lukunopeudesta, eikä mitään tietoa voida sen avulla tarjota suoraan nopeasta muistista. [36; 16, s. 535.] Näihin ongelmiin on kehitetty moduuleja, jotka lisäävät nämä toiminnallisuudet Drupaliin, ja niiden avulla voidaan rakentaa suuriakin käyttäjämääriä kestävä Drupal-sivusto.

Kuvio 5 osoittaa havainnollisella tavalla, millä tasoilla kukin välimuisti ja sovellus toimivat suhteessa käyttäjään. Tässä esimerkkiasennuksessa on Drupalin lisäksi asennettu Pressflow, Varnish, Memcache ja Boost tai Authcache riippuen siitä, onko sivustolla pääasiallisesti vain anonyymejä käyttäjiä vai anonyymejä ja kirjautuneita käyttäjiä. Nämä sovellukset ja moduulit tarjoavat Drupalin tiettyihin osa-alueisiin suorituskykyparannuksia, mutta eivät suinkaan ole välttämättömiä suorituskykyisen verkkosivun tekemiseksi Drupalilla. Lähtökohtaisesti kannattaa optimointi aloittaa itse sovelluksen

toiminnan parantamisesta ja nopeuttamisesta ja vasta sen jälkeen ryhtyä harkitsemaan kolmannen osapuolen suorituskykyä parantavien sovellusten asentamista. [35.]

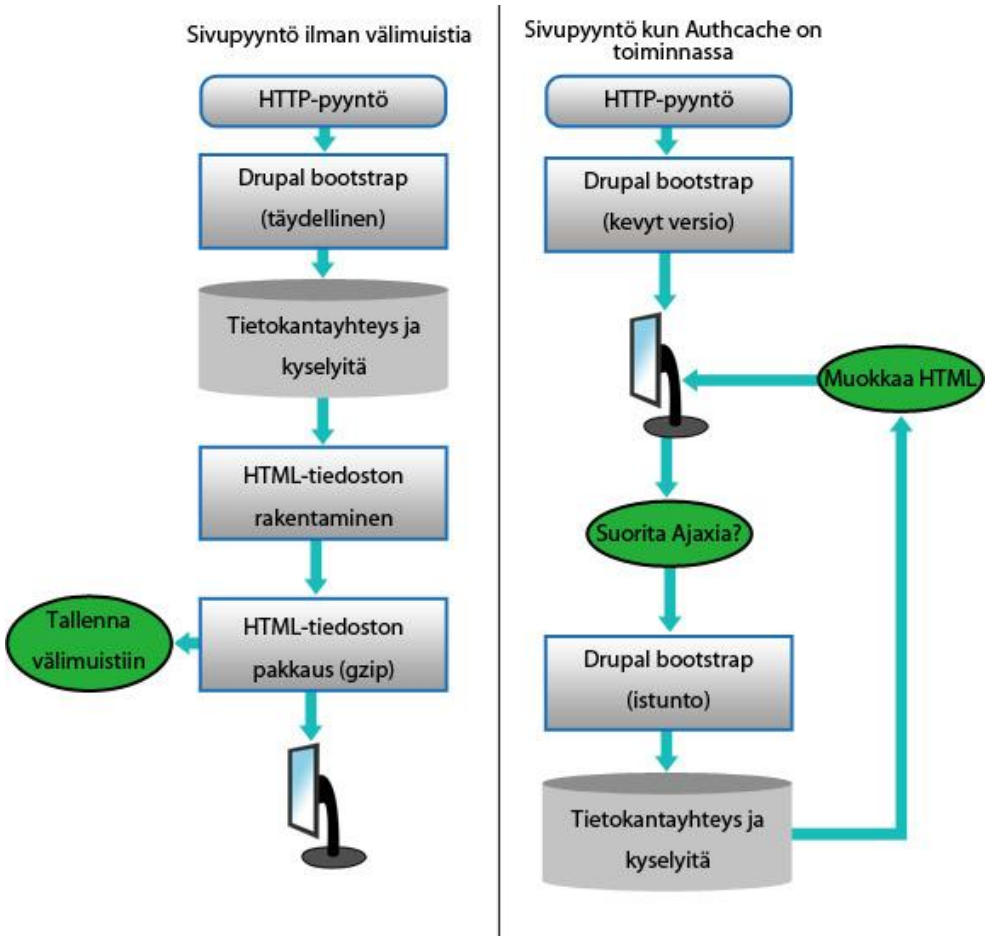


Kuvio 5. Välimuistitasot [35].

Authcache-moduuli

Authcache on Drupalin moduuli, joka mahdollistaa sisällön tarjoamisen välimuistista niin anonyymeille kuin kirjautuneille käyttäjille. Tämä moduuli tallentaa valmiiksi tehdyn HTML-sivun ja tarjoaa sen ilman tietokantakyselyjä tai muita palvelimelta vaadittavia suoritteita suoraan käyttäjille sellaisenaan. Sivut tallennetaan käyttäjäprofiilikohtaisesti, eli jokaiselle eri käyttäjätyypille tallennetaan oma versionsa esimerkiksi etusivusta. Näin on mahdollista näyttää eri käyttäjäprofiileille erilaiset versiot sivustosta, mutta samalla tallentaa sivu välimuistiin. [36.]

Authcache lähettää pyynnön palvelimelle sivuston latauduttua selaimessa. Tässä pyynnössä käynnistetään Drupalin *bootstrapin* kevyempi versio, jonka avulla on mahdollista suorittaa kyselyitä tietokantaan. Drupalin bootstrap on järjestelmän ydintoimintoja ja vakioarvoja sisältävä tiedosto, joka suoritetaan heti alussa, kun järjestelmä ladataan [37]. Authcache ei kuitenkaan tarvitse kaikkia sen sisältämiä toimintoja ja vakioita, joten moduulin kehittäjät ovat tehneet siitä kevennetyn version suorituskykyä ajatellen. Kevyemmän bootstrapin avulla suoritetut tietokantakyselyiden tulokset Authcache lisää sivustolle Ajaxilla, ja tällöin on mahdollista päivittää käyttäjälle yksilöllisiä tietoja, vaikka itse sivusto onkin ladattu välimuistista. Tämä nopeuttaa kirjautuneen käyttäjän saamaa palautetta sivustolta, koska Authcache pystyy tarjoamaan vasteen käyttäjän toimille 1–2 millisekunnissa. Ilman Authcachea kirjautuneen käyttäjän on odotettava koko sivupyyntö alusta saakka, jolloin vastauksen saaminen voi kestää jopa 200 millisekuntia. Kuviosta 6 selviää Authcachen yksinkertaistettu toimintaperiaate eli staattisen sivun latautumisen jälkeen tapahtuvat toiminnot. [36.]



Kuvio 6. Authcachen toimintaperiaate [36].

Authcache voi käyttää kahdentyyppistä välimuistia. Oletusarvoisesti se käyttää Drupalin tietokantaa tiedon tallentamiseen, mutta moduulin tekijät suosittelevat käyttämään Authcachen kanssa moduulia, kuten Memcache, joka käyttää tiedon varastointiin nopeaa keskusmuistia eikä tietokantaa. Tällöin välimuisti saadaan toimimaan nopeammin eikä tietokantaa kuormiteta välimuistin kyselyillä. [36; 16, s. 535.]

Boost-moduuli

Boost on Drupalin moduuli, joka tarjoaa helpon tavan lisätä staattinen sivuvälimuisti Drupal-sivustolle. Authcachesta poiketen Boost ei paranna sivuston suorituskykyä millään tavoin sisään kirjautuneille käyttäjille, joten sen asentamista kannattaa harkita vain siinä tapauksessa, että suurin osa sivuston käyttäjistä on anonyymejä käyttäjiä. [38.]

Boost-moduulin toimintaperiaatteena on ohittaa mahdollisimman paljon välikäsiä pyyntöihin vastattaessa, eli käytännössä moduuli ohittaa Drupalin ja tietokannan täysin ja tarjoaa suoraan staattisen HTML-sivun käyttäjälle palvelimen kiintolevyltä. Parhaimmassa tapauksessa selain tukee gzip-pakkausta, jolloin Boost tarjoaa selaimelle HTML-sivun, XML-, CSS- ja JavaScript-tiedostot pakattuina versoina minimoiden näin latausajan. [38.]

Moduulissa on myös huomioitu välimuistin heikkous eli se, että ennen kuin käyttäjä on kerran käynyt tietyllä sivulla, se ei ole välimuistissa. Boost sisältää sisäänrakennetun *crawlerin* eli hieman hakurobottia muistuttavan ohjelman, joka käy sivustoa läpi automaattisesti. Tällöin crawlerin läpikäymät sivut tallentuvat välimuistiin ja ne voidaan tarjota käyttäjälle nopeasti ilman, että yhdenkään anonyymin käyttäjän on tarvinnut ladata niitä hitaammalla tavalla. [35; 38.]

Memcache-moduuli

Memcache on Drupalin moduuli, joka mahdollistaa sivujen tarjoamisen käyttäjälle nopeasta keskusmuistista hitaamman tietokantayhteyden sijaan. Sovelluksen tärkein toiminnallisuus on siis sen kyky kirjoittaa ja lukea tietoa nopeasta keskusmuistista. Keskusmuistiin tallennetun tiedon on kuitenkin oltava sellaista, että sen häviämisellä ei ole

suurta merkitystä, ja juuri kaiken tällaisen tiedon Memcache pyrkii tarjoamaan nopeinta mahdollista reittiä käyttäjälle. Luonnollisesti kun tietokanta poistetaan tiedon välittämisketjusta, sen kuorma vähenee, ja tämän takia Memcache on hyvä työkalu varsinkin sivustoille, joissa tietokanta on suurin pullonkaula suorituskyvyssä. [16, s. 535.]

Memcache toimii siten, että se tallentaa keskusmuistiin objektit, joille se antaa yksilölliset tunnisteet. Käyttäen näitä tunnisteita se kutsuu pyydettyä objektia muistista. Memcache ei välitä saamansa tiedon tyypistä millään tavoin, vaan se kohtelee kaikkea tietoa samalla tavalla, joten se on kuin lyhytaikainen muisti sovellukselle. [16, s. 535–536; 39.]

Memcache ei ole aivan yhtä yksinkertainen asentaa kuin muut mainitut suorituskykyä parantavat moduulit, ja se tarvitsee itse moduulin lisäksi PECL-laajennuksen asentamisen palvelimelle. Tämä laajennus lisää PHP:lle tuen Memcachen vaatimia toiminnallisuuksia varten. Hankalammasta asennuksesta huolimatta Memcachen tarjoama hyöty on suuri, ja se on ohjelmoitavissa monipuolisesti. Sivuston ylläpitäjät voivat itse päättää, mitä moduuli tallentaa välimuistiin ja näin keskittyä juuri niihin ongelmiin, jotka aiheuttavat suurimmat suorituskykyongelmat sivustolla. [16, s. 535–536.]

Vaikka Memcache onkin Drupalin moduuli, sitä voidaan käyttää myös muissa sovelluksissa ja se on yleisluontoinen toimintansa suhteen. Sen toiminta ei myöskään sijaitse aivan täysin sovellustasolla, vaan suuri osa toiminnasta on palvelimella tapahtuvaa tietokannan kuorman vähentämistä ja pyyntöihin vastaamisen nopeuttamista. Huolimatta sijainnista sovellustason ja palvelintason välillä sen toiminta nopeuttaa sovelluksen sisällä olevien muiden moduulien toimintaa ja tarjoaa niille nopeamman vasteen perinteiseen tietokantakyselyyn verrattuna. [36; 39.]

3.4 Pressflow-sovellus

Pressflow on Drupalin ytimen pohjalta tehty vapaan lähdekoodin ohjelma, joka on erityisesti suunniteltu sivustoille, jossa on suuret kävijämäärät. Se pohjautuu Drupalin ytimen parannuksiin, joilla mahdollistetaan parempi suorituskyky, skaalautuvuus ja käänteisen välityspalvelimen käyttö. Useat Pressflowssa mukana olevat muutokset on otettu käyttöön seuraavassa Drupalin versiossa. Tästä hyvänä esimerkkinä on nyt uusin

Drupal-versio 7, jonka sisältämiä suorituskykyparannuksia Pressflow tarjosi jo ennen seitsemännen version julkaisua Drupalin kuudenteen versioon. [40; 41.]

Drupalin skaalautuvuutta on parannettu Pressflow'ssa lisäämällä mahdollisuus monistaa sivuston tietokanta useille palvelimille. Tämä tuo lisää nopeutta tietokantakyselyihin, jotka ovat usein kaikista suurin pullonkaula verkkosivuilla. Tietokannan monistaminen nopeuttaa sivuston toimintaa siten, että isäntäpalvelimen kuormaa vähennetään siirtämällä osa raskaista kyselyistä orjapalvelimille, joissa on sivuston tietokanta monistettuna. Tämä tekniikka on käytössä suurilla Drupal-sivustoilla, kuten drupal.org. Toinen merkittävä tietokantaparannus Pressflow'ssa on useiden eri tietokantatyyppien tuen karsiminen. Drupalin perusasennus tukee eri tietokantatyyppejä, mutta Pressflow on optimoitu siten, että se tukee vain eniten käytettyä MySQL-tietokantaa. Näin ollen tietokannan toiminta voidaan optimoida nopeammaksi kuin Drupalin perusasennus, koska muiden tietokantatyyppien toiminta voidaan unohtaa ja optimoida kyselyt toimimaan vain MySQL-tietokannan kanssa.

Pressflow lisää Drupaliin toiminnallisuuden, joka kertoo välityspalvelimelle, mitä sisältöä voidaan tallentaa välimuistiin ja mitä ei. Tämä toiminnallisuus mahdollistaa käänteisen välityspalvelimen, kuten Varnish tai Squid, käytön, ja se on yksi tärkeimmistä palvelimen kuormaa vähentävistä toimenpiteistä, kun sivustolla on suuri määrä käyttäjiä. Käänteisen välityspalvelimen tuen lisäksi Pressflow on optimoitu käyttämään vain PHP:n viidettä versiota, toisin kuin Drupalin kuudennen version perusasennus. Uudemman PHP-version funktiot toimivat nopeammin kuin Drupalin sisältämät vanhemman version funktiot, mikä taas lisää koodin toiminnan nopeutta. [41.]

3.5 Sovelluksen ulkopuolinen välimuisti

Drupal-sisällönhallintajärjestelmän versiosta 6 on monille jäänyt hidas vaikutelma. Hitauteen vaikuttavat monet seikat, mutta Drupalin yksinkertainen laajennettavuus ja monipuolisuus vaikuttavat tähän varmasti eniten. Moduulien helppo asentaminen johtaa helposti sivustoon täynnä tarpeettomiakin moduuleja, jotka syövät itse sivuston suorituskykyä. [17.] Uuden Drupal-asennuksen on kuitenkin todettu olevan nopeampi kuin esimerkiksi kilpailevan sisällönhallintajärjestelmän, Joomlan, jota yleisesti pidetään vähemmän monipuolisena ja vaikeammin skaalattavana verkkosivualustana [42]. Monista eri mielipiteistä huolimatta voidaan kuitenkin todeta, että Drupalin kuudes versio

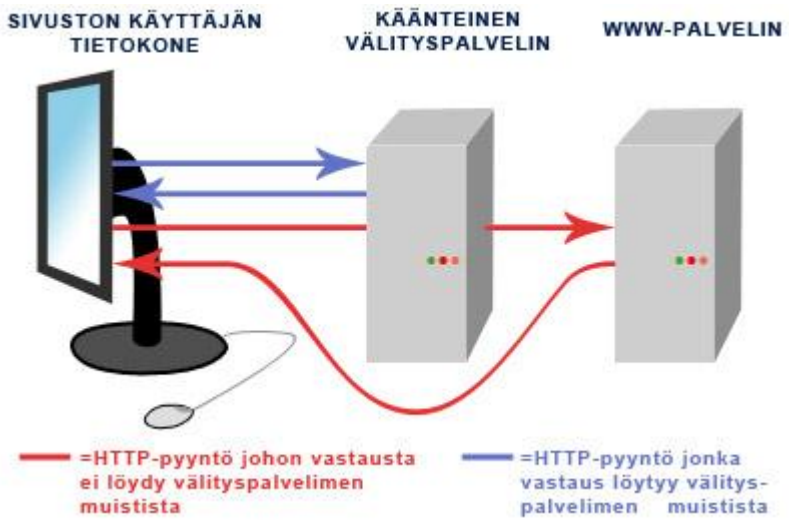
on osaltaan puutteellinen suorituskyvyn parantamisen osalta. Tästä kertoo esimerkiksi käänteisen välityspalvelimen tuen puute.

Drupalin kuudennen version suorituskyvyn parantamiseksi yritys nimeltä Four Kitchens on tehnyt Drupalin ytimestä oman versionsa nimeltään Pressflow. Pressflow ei kuitenkaan yksin riitä suorituskyvyn parantamiseen, vaan se vain mahdollistaa Drupalin toiminnan muiden suorituskykyä parantavien sovellusten kanssa. Yksi esimerkki tällaisesta ohjelmasta on Varnish. [40; 41.]

Käänteinen välityspalvelin

Käänteinen välityspalvelin on sovellus, joka tallentaa mahdollisimman paljon tietoa nopeaan RAM-muistiin, josta se tarjotaan suoraan käyttäjälle. Se voi olla fyysinen palvelin tai sovellus, joka toimii www-palvelimella. [43; 44.]

Kuten kuviosta 7 käy ilmi, käänteisellä välityspalvelimella tarkoitetaan käyttäjän ja tavallisen www-palvelimen väliin tulevaa välityspalvelinta, joka näyttäytyy käyttäjälle aivan kuin se olisi tavallinen www-palvelin.



Kuvio 7. Käänteisen välityspalvelimen toimintaperiaate.

Käänteistä välityspalvelinta voidaan käyttää välimuistin tarjoavana palvelimena ennen varsinaista www-palvelinta. Käänteinen välityspalvelin päättää itse, mihin se lähettää saapuneet pyynnöt käyttäjältä, jolloin staattinen välityspalvelimen muistiin tallennettu

sisältö voidaan tarjota suoraan välityspalvelimelta ilman, että tarvitaan pyyntöä itse www-palvelimelta. Sisältöä, jota välityspalvelimella ei ole tarjota, ohjataan edelleen www-palvelimelle, mutta koska kaikki pyynnöt eivät enää ohjaudu suoraan www-palvelimelle, sen kuormitus pienenee. [40; 43.] Välityspalvelimen toiminta eroaa siis www-palvelimen toiminnasta siten, että välityspalvelin tarjoaa vain suoraan muististaan löytyviä vastauksia, kun taas www-palvelin suorittaa pyyntöihin liittyvät prosessit ja tietokantahaut.

Varnish-kuormantasaajasovellus

Varnish on vapaaseen lähdekoodiin perustuva kuormantasaaja. Jos se on käytössä verkkosivulla, se on ensimmäinen palvelin, johon selain ottaa yhteyden eli se toimii käyttäjien ja www-palvelimen välillä tasapainottamassa palvelimelle tulevaa kuormaa. Varnishia käytetään pääasiassa mahdollistamaan sivuston osien välimuistiin tallennus, mutta se voidaan ohjelmoida tekemään myös paljon muuta. Tähän ohjelmointiin käytetään VCL-kieltä, joka on lyhenne sanoista *Varnish Configuration Language*. Sen avulla voidaan esimerkiksi määrittää, mikä sisältö tallennetaan välityspalvelimen välimuistiin ja miten.

Välimuistina toimimisen lisäksi Varnish voidaan ohjelmoida ohjaamaan palvelimelle saapuva liikenne toiselle palvelimelle osoitteen mukaan tai muuttamaan kirjoitettu verkko-osoite haluttuun muotoon. Varnish toimii heti asennuksen jälkeen ilman ohjelmointia, mutta välimuistiin tallentuu vain pieni määrä tietoa ja ohjelma toimii muutenkin hyvin varovaisesti, joten on suositeltavaa, että sen asetukset käydään huolellisesti läpi ja sen toiminta optimoidaan parhaan lopputuloksen aikaansaamiseksi. [44; 45.]

4 Suorituskyvyn mittaustyökalut

Tärkeä osa sivuston optimointia on myös sen suorituskyvyn mittaus, jotta voidaan todentaa, onnistuiko optimointi. Jo silmämääräinen sivun latausnopeuden tarkastelu kertoo jotakin optimoinnin tasosta, mutta se ei kuitenkaan kerro sitä, mihin optimointi tulisi kohdistaa. Verkkosivun suorituskyvyn parantamiseksi täytyy siis sen nopeutta ja kuormansietokykyä pystyä testaamaan jollakin tavalla. Sivuston nopeuden testaaminen on sinällään yksinkertaista, mutta sivuston hitauden syyn selvittämiseen ilman valmiita työkaluja voi kulua paljon aikaa ja ilman testityökalun osoittamaa tietoa optimoinnista tulee helposti arvailua. Tämän ansiosta verkkosivua testattaessa on hyvä käyttää työkaluja, jotka mittaavat sivuston suorituskykyä ja kertovat parhaimmillaan, mistä sivuston hitaus mahdollisesti johtuu.

4.1 YSlow- ja Google Page Speed -työkalut

YSlow on Yahoon kehittämä työkalu sivuston suorituskyvyn tutkimiseen ja sen toiminta perustuu Yahoon sisäisen, suorituskykyyn keskittyvän tiimin määrittelemiin sääntöihin sivuston toiminnan nopeuttamiseksi. YSlow on yksi ehkä helpoiten käytettävistä ja tulkittavista suorituskykyä mittaavista sovelluksista. Muista sovelluksista poiketen se ei ole erillinen ohjelma, vaan Firefox-selaimen liitännäinen. [46.]

YSlow'n asentaminen on yksinkertaista ja käyttöönotto nopeaa. Aluksi Firefox-selaimeen tarvitsee asentaa toinen liitännäinen, nimeltä Firebug, jonka YSlow tarvitsee toimiakseen. Tämän jälkeen asennetaan itse YSlow-liitännäinen ja voidaan aloittaa suorituskykytestaus. Testaus suoritetaan yksinkertaisesti menemällä halutulle sivulle Firefox-selaimella, kytkemällä Firebug-liitännäinen päälle vasemmassa alakulmassa näkyvästä kuvakkeesta, siirtymällä YSlow-välilehteen ja päivittämällä sivu uudestaan. [46; 47.]

YSlow analysoi sivustoa käyttäen 22:ta sääntöä 34:stä säännöstä, jotka Yahoon suorituskykyasiantuntijat ovat laatineet, ja juuri nämä 22 sääntöä on valittu siitä yksinkertaisesta syystä, että ne pystytään testaamaan. Hyvänä esimerkkinä säännöistä on ensimmäinen sääntö, joka kuuluu seuraavasti: Minimoi HTTP-pyynnöt. Analysoidessaan sivustoa YSlow tarkistaa, kuinka tarkasti kutakin sääntöä noudatetaan sivulla. Säännön

rikkomisesta vähennetään pisteitä, ja lopuksi jokaiselle säännölle annetaan arvosana. Tämä arvosana kertoo, kuinka hyvin sivusto noudattaa sääntöä. Tämän jälkeen muodostetaan yleisarvosana sivun saamista yhteispisteistä, painottamalla pisteitä säännön tärkeyden mukaan. [47.]

Tulokseksi YSlow'n suorittamisen jälkeen saadaan ehdotuksia, joiden avulla sivuston suorituskykyä voitaisiin parantaa, statistiikkaa, sivuston komponenttien yhteenveto ja työkaluja sivuston suorituskyvyn parantamiseksi tai analysoimiseksi tarkemmin. Hyvänä esimerkkinä YSlow'n suosittelemista työkaluista on Smush.it-työkalu, joka löytyy YSlow'n työkalut-välilehdeltä. Se optimoi verkkosivulta löytyviä kuvia häviöttömillä pakkaustekniikoilla, jolloin kuvien laatu ei heikkene, mutta niiden koko pienenee. Kun suoritetaan tämä ohjelma kuville sivustolla, nähdään kuinka monta prosenttia kuvien tiedostokoko pienenee, jos käyttää Smush.it-prosessoituja kuvia sivustolla. Ohjelman suorittamisen jälkeen kuvat voidaan ladata pakatussa tiedostossa, josta ne voi vaihtaa sivustolla aiemmin olleiden kuvien tilalle. [46; 47; 48.]

Google Page Speed on hyvin paljon YSlow'ta muistuttava Googlen kehittämä sivustojen suorituskykyä analysoiva työkalu. Se on vapaaseen lähdekoodiin perustuva työkalu, joka voidaan asentaa Google Chrome -selaimen lisäosaksi tai sitä voidaan käyttää missä tahansa selaimessa osoitteessa http://pagespeed.googlelabs.com/. YSlow'n tapaan sekin testaa, kuinka hyvin sivusto noudattaa optimaalisen verkkosivun periaatteita, ja pisteittää sivuston niiden perusteella. Testaustoimintojen lisäksi Google Page Speed -palvelu tarjoaa Apache-ohjelmistoa käyttäville palvelimille moduulia, joka optimoi sivustoa ja sen tarjoamia resursseja kirjoittamalla ne uudelleen optimaalisemmin. [49.]

Google Page Speed -lisäosan asennus Chrome-selaimeen on yhtä helppoa ellei helpompaakin kuin YSlow'n asentaminen Firefox-selaimeen. Google Page Speed integroituu Chrome-selaimeen sisäänrakennetuun kehitystyökaluun, joten sen toimintavalmiuteen saattamiseen tarvitaan vain itse Page Speed -lisäosan asentaminen. Lisäosa on kuitenkin vasta kokeiluasteella, joten Chrome-selaimen asetuksia täytyy muuttaa siten, että selain sallii kokeellisten lisäosien asentamisen. [49; 50.]

4.2 Load Impact -työkalu

Load Impact on verkossa toimiva verkkosivujen kuormankestävyyden testausjärjestelmä. Sen avulla voidaan testata, kuinka paljon käyttäjiä sivusto kestää yhtäaikaisesti eli kuinka suuren kuorman sivusto aiheuttaa palvelimelle, kun tietty määrä virtuaalisia käyttäjiä käyttää sivustoa yhtäaikaisesti. Järjestelmä lisää virtuaalisten käyttäjien määrää sivustolla portaittain ja mittaa samalla sivuston latausaikoja ja tallentaa ne muistiin. Tästä datasta järjestelmä koostaa yhteenvedon ja graafisia käyriä, joita voidaan tulkita sivuston suorituskykyä halutulla käyttäjämäärällä valituilla sivuston osioilla. [51.]

Järjestelmän käyttö on maksullista, ja kalleimmilla lisensseillä on mahdollista simuloida kymmenien tuhansien yhtäaikaisten virtuaalisten käyttäjien luoma kuorma sivustolle. Käyttäjiä voidaan simuloida sivustolle ympäri maailmaa sijaitsevasta palvelinverkostosta, jossa on nopeilla yhteyksillä varustettuja tehokkaita palvelinklustereita. Käytännössä palvelulla on mahdollista kaataa palvelin tai palvelimet kuormittamalla niitä suuremmalla käyttäjämäärällä, kuin mitä ne pystyvät palvelemaan. [51.]

4.3 Siege-työkalu

Siege on hyvin samantyyppinen ohjelma kuin Load Impact, eli sen tarkoituksena on selvittää, kuinka hyvin palvelin kestää yhtäaikaisten käyttäjien aiheuttamaa kuormaa. Ohjelma perustuu vapaaseen lähdekoodiin, ja se kehitettiin kehittäjille, jotta he voisivat testata koodinsa toimivuutta monen käyttäjän kuorman alla Internetissä. Toimintaperiaate on myös sama kuin Load Impactissa, eli ohjelma simuloi halutun määrän virtuaalisia selaimia selaamaan verkkosivun tiettyä osaa ja tallentaa saamansa tulokset. Käyttöliittymä on kuitenkin täysin tekstipohjainen, joten tuloksien tulkitseminen ei ole aivan yhtä yksinkertaista kuin Load Impactia käytettäessä, vaan lukuja täytyy myös osata tulkita. [52.]

Ohjelman suorittamisen avulla saadaan sivustolle useita arvoja, joista voidaan nähdä, kestääkö se testattua käyttäjäkuormaa. Tärkein saatu luku on varmasti *availability* eli saavutettavuus, joka kertoo, onko sivusto ollut koko testin ajan virtuaalisten käyttäjien saavutettavissa. Toinen tärkeä luku on *concurrency* eli sivuston keskimääräinen yhtäaikaisten yhteyksien määrä. Tämä luku saadaan jakamalla jokaiseen yksittäiseen pyyn-

töön ja siihen saatuun vastaukseen kulunut aika kaiken kaikkiaan mittaukseen käytetyllä ajalla. Korkea concurrency-arvo kielii siitä, että palvelimen suorituskyky on kovilla eli se ei kestä yhtäaikaisista käyttäjistä aiheutunutta kuormaa. [53.]

Ohjelman käyttäminen on melko yksinkertaista. Komentorivillä ohjelmalle annetaan testin kohteena olevan sivuston osoite, yhtäaikaisten käyttäjien määrä ja testin kesto. Suoritettuaan testin ohjelma kirjoittaa komentoriville saadut mittaustulokset, kuten saavutettavuuden, keskimääräisen yhtäaikaisten yhteyksien määrän, tietoa testin aikana liikkuneen tiedon määrästä ja tapahtumien kestosta. Suorittamalla testi useampaan kertaa nousevalla määrällä yhtäaikaisia käyttäjiä saadaan suuntaa antava kuva siitä, kuinka paljon yhtäaikaisia käyttäjiä sivusto kestää. [53.]

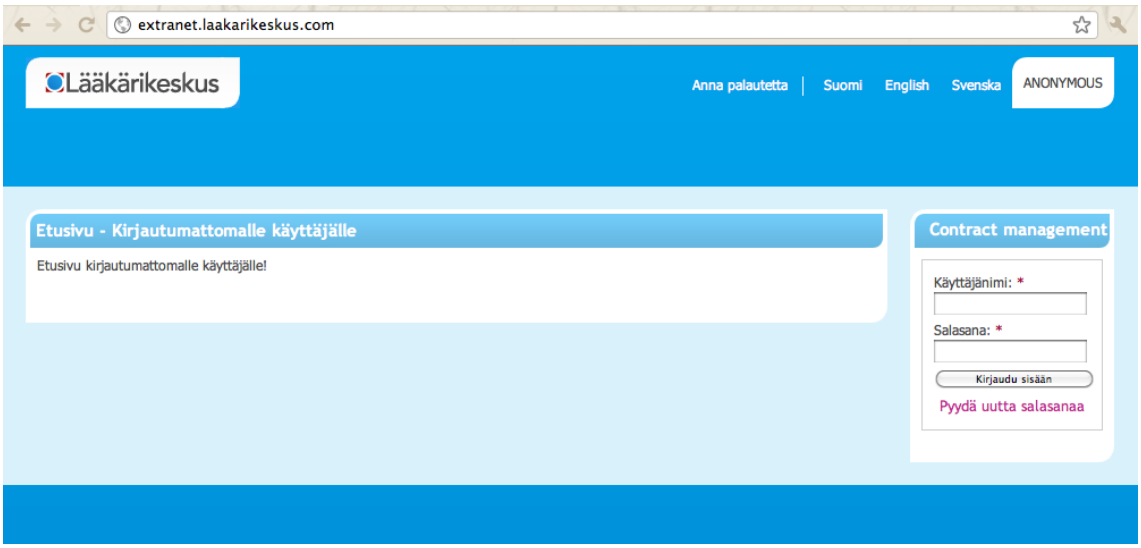
5 Verkkopalvelun optimointi

5.1 Verkkopalvelun optimointi Helsingin Lääkärikeskus Oy:lle

Insinöörityönä tehtiin uusi ekstranetverkkopalvelu Helsingin Lääkärikeskus Oy:lle, joka on yksi Suomen suurimpia yksityisiä terveys- ja hyvinvointipalveluita tarjoavia yrityksiä. Se perustettiin vuonna 1966, ja sillä on toimipisteitä Uudellamaalla. Yrityksen palveluihin lukeutuvat työterveydenhoitopalvelut ja erikoislääkäreiden palvelut, ja se tarjoaa myös erillisiin ihmisryhmiin, kuten nuoriin naisiin, kohdistuvia palveluita erikoistuneissa yksiköissään. [54.]

Uuden verkkopalvelun toiminta-ajatuksena on tarjota yrityksen asiakasyrityksille pääsy Internetin välityksellä raportteihin ja muihin asiakastietoihin suojatusti. Sivuston käyttäjät ovat siis aina sisään kirjautuneita käyttäjiä. Sivuston alustana on Drupal-sisällönhallintajärjestelmän kuudes versio, ja se noudattaa yrityksen julkisen sivuston graafista ilmettä. Sen teema rakennettiin julkisen sivuston teeman pohjalta, mikä ei luonnollisesti ollut optimoinnin kannalta ideaali ratkaisu, mutta kustannustehokkaampi. Sivuston toiminta haluttiin optimoida mahdollisimman hyvin alusta alkaen, ja siihen sovellettuja optimointitoimenpiteitä testattiin monilta osin ensimmäistä kertaa, sillä usein sivustot optimoidaan lähes ainoastaan anonyymien käyttäjien käyttökokemusta silmällä pitäen.

Kuten kuviosta 8 nähdään, verkkopalvelu vaikuttaa ulospäin melko yksinkertaiselta sovellukselta, mutta sen optimointi oli tavallista verkkosivua haastavampaa.



Kuvio 8. Helsingin Lääkärikeskus Oy:n verkkopalvelun etusivu.

Palvelun käyttötarkoituksen takia Drupal jouduttiin asettamaan tilaan, jossa tiedostojärjestelmä on *private* eli yksityinen. Tämä tarkoittaa sitä, että Drupal vaatii järjestelmään erikseen ladattuihin tiedostoihin käyttöoikeudet, eli ne eivät ole kaikkien ladattavissa [55]. Tämä taas asetti optimoinnille haasteita, koska esimerkiksi ytimen sisäänrakennettu CSS- ja JavaScript-tiedostojen optimointitoiminnallisuus tarvitsee kaikille käyttäjille vapaan pääsyn järjestelmään ladattuihin tiedostoihin. Tämä johtuu siitä, että järjestelmä tallentaa optimoidut tiedostot samaan kansioon kuin järjestelmään ladattavat, ulkopuoliset tiedostot ja tähän sisältöön vaaditaan käyttöoikeudet. [56.]

5.2 Sovelluksen lähtökohta

Helsingin Lääkärikeskus Oy:lle tehty ekstranetverkkopalvelu oli lähtökohtaisesti optimoimaton Drupal-asennus, jossa oli optimoimaton teema. Teema sisälsi useita CSS- ja JavaScript-tiedostoja, joita ei ollut optimoitu millään tavoin, vaan ne ladattiin yksitellen ja pakkaamattomina versioina jokaisen sivupyynnön aikana. Järjestelmään oli luotu asiakkaan tarvitsemat toiminnallisuudet, kuten kieliversiot ja ulkopuoleisten tiedostojen lataaminen vain tiettyjen käyttäjäryhmien saavutettaviksi. Sivustolle ei ollut asennettu minkäänlaisia suorituskykyä parantavia moduuleita, vaan vain toiminnallisuuksien toteuttamiseksi vaadittavia moduuleja ja järjestelmän hallintaa helpottavia työkaluja, kuten Administration menu -moduuli, joka lisää järjestelmän hallitsijalle paremman käyttöliittymän sovelluksen hallintaan, kuin mitä perusasennus tarjoaa.

Drupalin mukana tuleva sovelluksen välimuisti oli kytketty pois päältä, koska sovelluksen kehittämisvaiheessa jatkuva välimuistien tyhjentäminen muutosten havaitsemiseksi olisi rasittavaa. Sivujen pakkaus oli myös kytketty pois päältä suorituskykyasetuksista, jolloin kaikki tiedostot tulivat selaimen niitä pyytäessä täysikokoisina palvelimelta. Sovelluksen mukana tuleva CSS- ja JavaScript-tiedostojen pakkaus ei myöskään ollut päällä aiemmin mainitun private-tiedostojärjestelmän takia.

Sivusto sijaitsee Crescomin Oy:n optimoidulla palvelimella, jonka asetukset on määritelty suorituskykyisiksi ja palvelemaan mahdollisimman hyvin sillä sijaitsevien sivustojen tarpeita. Tämän vuoksi palvelimelle on asennettu myös Varnish, joka tallentaa välimuistiin testattavalla sivustolla esiintyviä kuvia ja erillisiä tiedostoja, kuten CSS- ja JavaScript-tiedostoja. Varnish ei kuitenkaan tallentanut välimuistiinsa sivuston sisältöä, koska sivusto toimii Drupalin perusversiolla eikä Pressflow'lla. [56.]

5.3 Lähtötilanteen mittaaminen

Verkkosivuston suorituskyvyn mittaamiseen käytin neljää eri ohjelmistoa, jotka mittasivat sivustoa kahdella eri tavalla. Valitsin YSlow- ja Google Page Speed -työkalut järjestelmän suorituskykyisyyden mittaamiseen ja Load Impactin sekä Siegen mittaamaan sovelluksen ja palvelimen yhteistoiminnan suorituskykyisyyttä. Kahta viimeksi mainittua verkkosivuja virtuaalikäyttäjillä kuormittavaa testausohjelmistoa päätin käyttää siitä huolimatta, että optimointi keskittyi pelkästään sovelluspuolelle, koska hyvin optimoitu sovellus ei kuormita palvelinta yhtä paljon kuin optimoimaton versio. Näin ollen oli mahdollista nähdä eroja tuloksissa tällaisillakin testausmenetelmillä ja jopa havaita mahdollisia pullonkauloja suorituskyvyssä. [56.]

Load Impact vaati muutamien asetusten tarkistamista ennen testin aloittamista. Alkuun täytyi valita testikäyttäjien sivustolla kulkema reitti, jonka ohjelma tallensi ja suoritti jokaisella ajolla. Tallentamani reitti kulki siten, että aluksi käyttäjä latasi etusivun selaimeensa ja kirjautui järjestelmään luomallani testitunnuksella. Tämän jälkeen käyttäjä painoi linkkiä osioon, jossa listataan kaikki käyttäjän käytettävissä olevat raportit, minkä jälkeen sivusto suljettiin. Näin saatiin kolme erityyppistä sivuston osiota testattavaksi, eli etusivu kirjautuneelle ja kirjautumattomalle käyttäjälle sekä listaussivu. Testin tyypiksi valittiin *ramp-up*, jolla tarkoitetaan sitä, että ohjelma kuormittaa sivustoa as-

teittain, nousevalla määrällä käyttäjiä. Testaus aloitettiin kymmenellä yhtäaikaisella käyttäjällä ja päätettiin 250 yhtäaikaiseen käyttäjään. Käyttäjien määrän nousu asetettiin kymmeneen käyttäjään, eli ohjelmisto kuormitti sivustoja jokaisen suorituskerran jälkeen kymmenellä käyttäjällä edelliskertaa enemmän. Testin nopeudeksi asetin toiseksi nopeimman vaihtoehdon, koska palvelimella oli samanaikaisesti muita julkisia sivustoja, joten sitä ei voitu kuormittaa liian pitkään. Jos palvelimella ollutta sivustoa olisi testattu pitkään, palvelimen muita sivustoja käyttävät käyttäjät olisivat häiriintyneet testauksen aiheuttamasta toiminnan hidastumisesta. Sovellusta kuormittavaa palvelinta ei voitu muuttaa, koska käytössä ollut lisenssi oli rajoitettu vain Tukholmasta toimivan rasitustestauspalvelimen käyttöön.

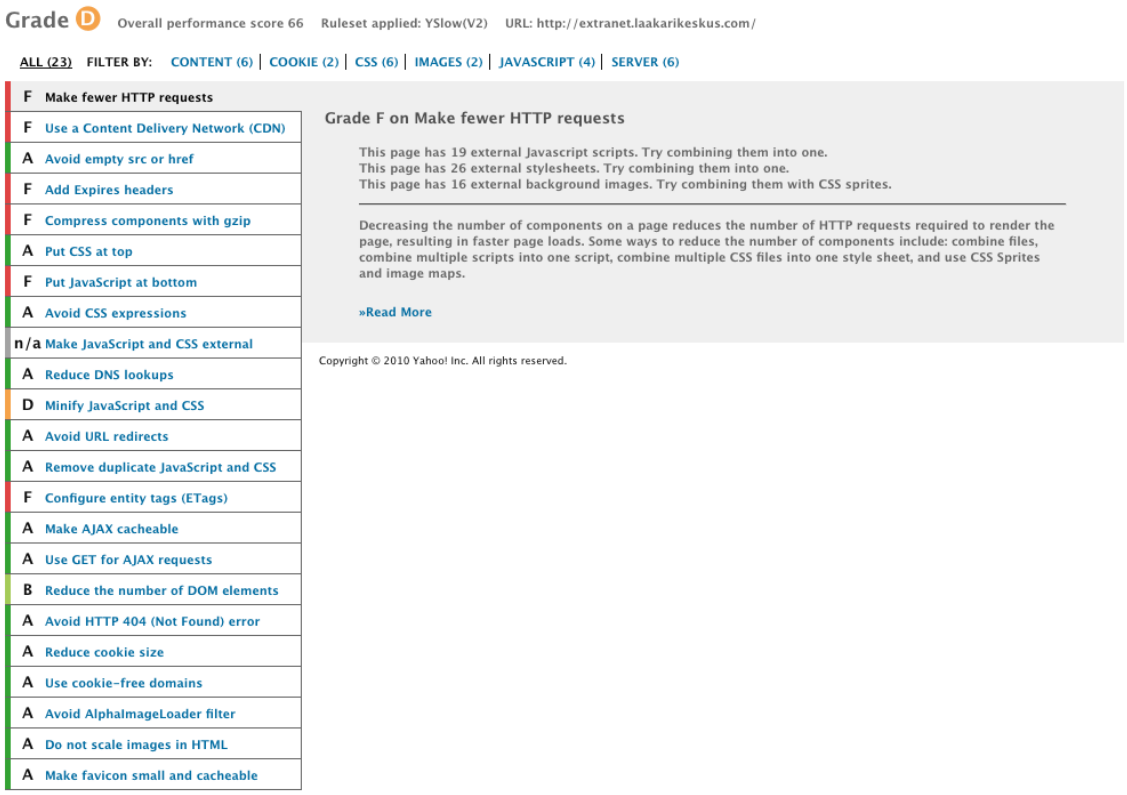
YSlow ja Google Page Speed eivät tarvinneet asennuksen jälkeen minkäänlaista asetusten määrittelyä, vaan testaaminen voitiin aloittaa yksinkertaisesti painamalla testi käyntiin napista. Siegen käyttäminen oli myös hyvin yksinkertaista, eli ohjelma käynnistettiin ja sille annettiin seuraavat tiedot: yhtäaikaisten käyttäjien määrä, testin kestoaika ja testattavan sivuston osoite.

Lähtötilanteen mittaustulokset

Ensimmäisenä suorituskyvyn mittaustoimenpiteenä sivusto testattiin YSlow'lla. Testi suoritettiin kirjautumattomana käyttäjänä etusivulla. Yleisarvosanaksi optimoimaton sivusto sai D:n asteikolla F–A, jossa A oli paras ja kokonaispisteet jäivät 66 pisteeseen sadasta. Käytin YSlow'n versiota 2 mittausten pohjana, eli sivuston toimintaa arvioitiin 23 eri säännön pohjalta.

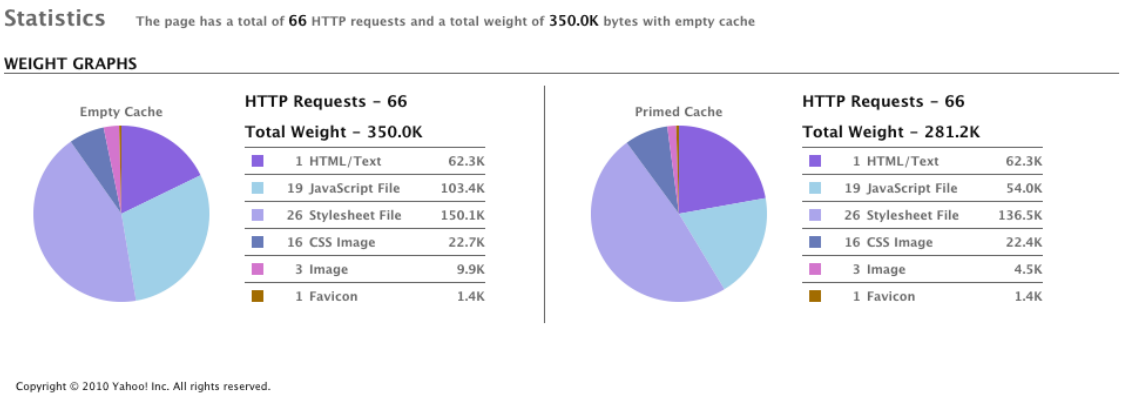
Mittaustulokset kuviossa 9 kertovat, että sivuston suurimmat puutteet ovat suuri HTTP-pyyntöjen määrä, tiedostojen otsikot, joille ei ole määritelty vanhenemisaikaa, tiedostojen pakkaamattomuus ja JavaScript-tiedostojen lisääminen sivustolle HTML-tiedoston alussa. CDN-järjestelmän käyttämättömyys sivustolla on myös luokiteltu huonoksi suorituskyvyn kannalta, mutta näin pienessä, Etelä-Suomeen keskittyvässä projektissa järjestelmän käyttäminen olisi ylitseampuvaa. Toisen suorituskykyä heikentäväksi luokitellun ominaisuuden sivustolla katsottiin olevan ETagien asetusten säätämisen puute, mutta tämä ominaisuus on järkevä vain siinä tapauksessa, että käytetään useaa palvelinta saman sivuston tarjoamiseen käyttäjälle. Tiedostoille voidaan näin antaa tunniste,

jonka selain lukee. Jos ETagit ovat määrittelemättä tarkemmin, kahdelta eri palvelimelta tarjotaan samaa tiedostoa eri ETageilla. Tämä tarkoittaa sitä, että vaikka selain olisi jo kerran ladannut tiedoston välimuistiinsa palvelimelta A, palvelimen B tarjoama sama tiedosto ladataan uudestaan, jos sen ETag poikkeaa alun perin palvelimelta A ladatun tiedoston ETagista. Lääkärikeskukselle tehty ekstranetverkkopalvelu tarjoillaan kuitenkin vain yhdeltä palvelimelta, eli tämä ominaisuus on sivuston suorituskyvyn kannalta turha. [57.]



Kuvio 9. YSlow'n mittaustulokset optimoimattomalle sivustolle.

Kuvio 10 kertoo selkeästi, kuinka huonosti sivusto tallentui selaimen välimuistiin. Vasemmalla kuvassa on sivustolta ladattavat osat välimuistin ollessa tyhjä, eli ensimmäistä kertaa sivustolle tultaessa, ja oikealla toinen sivun lataus, jolloin selaimen välimuistiin on tallennettu kaikki mahdollinen. Kuviosta kuitenkin nähdään, että selain lataa lähes kaikki elementit uudestaan palvelimelta ja HTTP-pyyntöjä tulee molemmilla kerroilla 66.



Kuvio 10. YSlow'n keräämät tiedot sivuston toiminnasta selaimessa.

YSlow'n lisäksi testasin sivuston myös Google Page Speed -lisäosalla, joka antoi optimoimattomalle sivulle murskaavan arvosanan 30/100. Googlen kehittämä työkalu ei tarjonnut aivan yhtä kattavasti tietoa kuin YSlow, mutta parannusta kaipaavat sivuston toiminnot kävivät selville myös sen antamista tuloksista. Google Page Speed erosi myös siinä YSlowsta, että se sovelsi sivustoon 30:tä sääntöä YSlow'n 23 säännön sijaan.

Kuten kuviosta 11 nähdään, eniten suorituskykyä heikentävinä ominaisuuksina Google Page Speed piti CSS-tiedostojen suurta määrää ja yleisesti sivuston pakkaamattomuutta. Rivien välistä voidaan lukea myös, että Googlen työkalu kehottaa YSlow'n tapaan vähentämään HTTP-pyyntöjen määrää yhdistämällä tiedostoja ja käyttämällä sprite-grafiikkaa. Selaimen välimuistin huono hyödyntäminen on myös huomion arvoinen suorituskykyä heikentävä seikka Google Page Speedin mukaan.

Kuvio 11. Google Page Speed -lisäosan antamat tulokset.

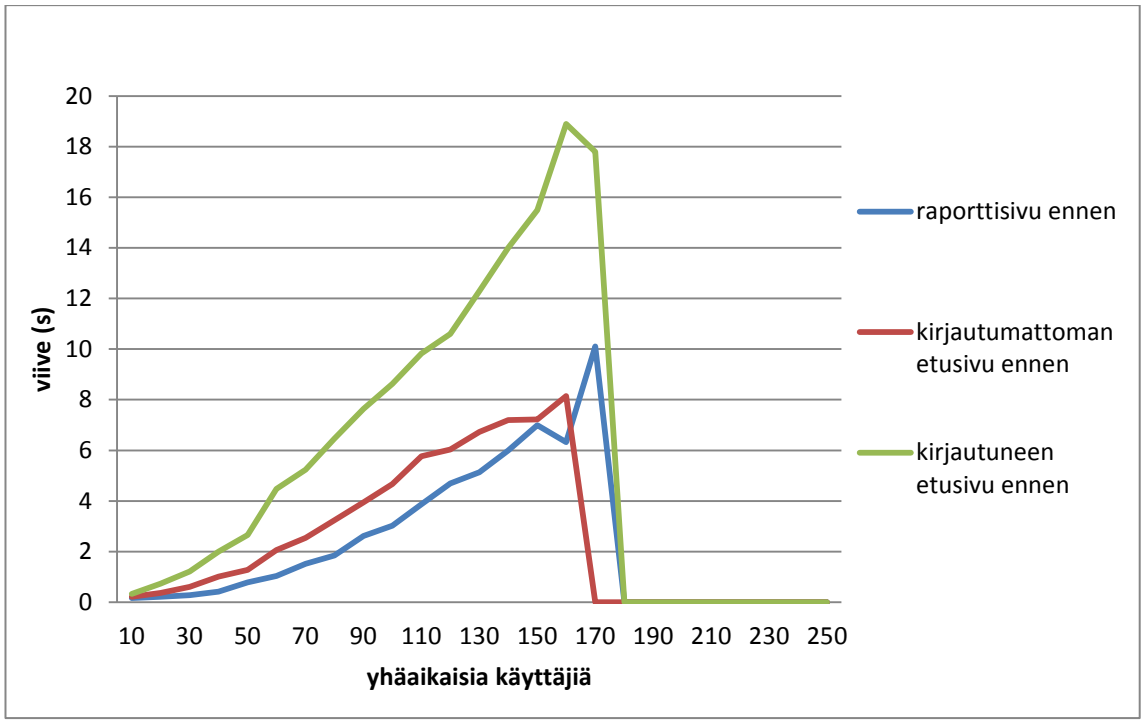
Ensimmäisessä Load Impactillä suoritetussa mittauksessa ennen optimointia palvelin kaatui kesken testin, ja siksi testiä ei voitu suorittaa loppuun. Palvelimen kaatuminen rasitustestissä kertoi jo siitä, että sivusto ei optimoimattomana kestä kovinkaan suurta käyttäjäkuormaa.

Kuviosta 12 nähdään selvästi, missä kohtaa palvelin kaatui, eli 160 yhtäaikaisen käyttäjän jälkeen rasitustestin aiheuttama kuorma oli liikaa palvelimelle. Kuvion kuvaaja kasvaa eksponentiaalisesti, eli siitä nähdään, että testi pystyi kuormittamaan palvelinta siinä määrin, että oli mahdollista arvioida, kuinka suurta määrää käyttäjiä palvelu kesti yhtäaikaisesti. Sivusto ei kuitenkaan kestänyt juuri yhtään yhtäaikaisia käyttäjiä testin mukaan, jos asetetaan hyväksyttäväksi viiveeksi sekunti, sillä kuvaaja alkaa nousun melko tasaisesti heti testin alkupäästä. Täytyy kuitenkin muistaa, että testiin kuului kolme vaihetta: kirjautumattomalle käyttäjälle näkyvä etusivu, kirjautuneelle näkyvä etusivu ja raporttien listaussivu. Iideaalisia yhden sekunnin kuvaajia näin monella sivulla on vaikea saavuttaa, joten hyväksyttävä viive tällaisella sivumäärällä olisi järkevämpi asettaa hieman ylemmäs, kuten 6–7 sekuntiin. Kuvaajasta nähdään, että sivusto kestäisi noin 55 yhtäaikaista käyttäjää, kun hyväksyttävä keksimääräinen viive on kuuden

ja seitsemän sekunnin luokkaa. [58.] Testin tulokset osoittavat, että vaikka palvelun viiveet olisivatkin riittävän alhaiset projektiin, kannattaa järjestelmässä suorittaa ainakin jonkinasteista optimointia, koska palvelin saatiin kaatumaan melko pienellä määrällä yhtäaikaisia käyttäjiä.

Kuvio 12. Sivuston viive ennen optimointia.

Load Impactin tuottamasta datasta on mahdollista myös selvittää yksittäisten sivujen viiveet, ja kuviossa 13 näkyvät suorittamani testin tulokset. Käyristä nähdään, että suurimman viiveen aiheutti kirjautuneen käyttäjän etusivu ja raporttisivu sekä kirjautumattoman käyttäjän etusivu latautuivat lähes puolet nopeammin kuin se. Käyrästä voidaan kuitenkin päätellä, että kirjautuneen ja kirjautumattoman käyttäjän välimuistin toiminnassa on selkeästi eroa ja että Panels-moduuli, jonka avulla kirjautuneen ja kirjautumattoman käyttäjän etusivu on toteutettu, hidasti sivuston toimintaa. Tämän voi päätellä siitä, että sinisellä merkitty raporttisivu latautui lähes yhtä nopeasti kuin punaisella korostettu kirjautumattoman käyttäjän etusivu. Käyrät nimittäin seuraavat toisiaan, vaikka raporttisivulla käyttäjä onkin kirjautuneena sisään ja välimuistin käyttö on lähes olematonta. Jos raporttisivulla olisi myös käytetty Panels-moduulia, käyrä seuraisi oletettavasti kirjautuneen käyttäjän etusivun vihreätä käyrää.



Kuvio 13. Yksittäisten sivujen viive ennen optimointia.

Viimeiseksi testasin sivuston suorituskykyä Siegellä. Kuormitin sovellusta aluksi 10:llä, sitten 12:lla ja viimeiseksi 50 yhtäaikaisella käyttäjällä. Testauksen kohteena toimi jälleen kirjautumattoman käyttäjän etusivu siitä syystä, että kirjautuneen käyttäjän etusivun kuormittaminen olisi ollut hyvin monimutkaista tällä ohjelmistolla.

Rasitustestin tulokset liitteessä 1 osoittavat sivuston vasteajan olleen keskimäärin 0,75 sekuntia, kun yhtäaikaisia käyttäjiä sivustolla oli 10. Otin tavoitteeksi selvittää, kuinka monta yhtäaikaista käyttäjää sivulla voi olla keskimääräisen latausajan ollessa noin yhden sekunnin luokkaa. Seuraavassa testiajossa sain tulokseksi 0,97 sekunnin latausajan 12 yhtäaikaisella käyttäjällä, eli voitiin todeta sivun kestävän optimoimattomana 12 yhtäaikaista käyttäjää ilman, että latausajat nousevat liian suureksi. Nostin vielä käyttäjämäärän 50:een testatakseni, miten nopeasti sivusto vastasi tällä määrällä käyttäjiä, ja sain tulokseksi 4,83 sekuntia, joka menee reilusti yli hyväksyttävän rajan. Sivuston keskimääräinen yhtäaikaisten yhteyksien määrä oli myös 50 yhtäaikaisella käyttäjällä melko korkea, eli 41,38, mikä kertoo siitä, että palvelimella alkoi käyttäjämäärien noustessa tätä suurempiin lukemiin olla vaikeuksia suorituskyvyn kanssa. [59.]

5.4 Optimointitoimenpiteet

Aloitin optimoinnin asentamalla Drupalin perusasennuksen sijasta Pressflow'n, koska palvelimella oli jo Varnish. Tällöin sain otettua hyödyn irti käänteisen välityspalvelimen ominaisuuksista käyttäjille, jotka eivät ole kirjautuneena sisään. Pressflow'n avulla myös Drupalin ytimen toiminta saatiin optimaalisemmaksi ja turha PHP:n neljännen version tuki poistui hidastamasta toimintoja.

Optimoimaton teema oli yksi syy huonoihin tuloksiin YSlow- ja Google Page Speed -testeissä. Teeman optimoimiseksi kaikki mahdolliset sivuston ulkoasuun liittyvät kuvat kuten, logo ja sivun elementtien taustakuvat, tallennettiin yhdeksi kuvaksi, sprite-grafiikaksi, josta ne poimittiin sivuston eri kohtiin tyylimäärittelyillä. Näin saatiin kahdeksan HTTP-pyyntöä vähennettyä yhdeksi pyynnöksi. Kaikkia kuvia ei kuitenkaan voitu yhdistää, koska jos taustakuvaa käytetään itse kuvaa suuremmassa elementissä, sprite-grafiikassa olevat muut kuvat tulevat näkyviin. Kuvatiedostojen optimoinnin lisäksi teeman CSS-tiedostot oli käytävä läpi. Niissä oli päällekkäisyyksiä ja turhia määrityksiä, koska teema oli rakennettu vanhan teeman päälle. Kaikkia näitä määrityksiä en määritysten suuren määrän ja rajallisen ajan takia saanut poistettua, mutta sain tiedostoista kuitenkin kooltaan pienempiä ja pystyin yhdistämään kaksi tiedostoa. Poistin myös ylimääräiset kommentit tiedostokoon optimoimiseksi.

Asensin sovellukseen Advanced CSS/JS Aggregation -moduulin, jonka avulla on mahdollista tallentaa optimoidut tiedostot eri kansioon, kuin mihin järjestelmän sisäänrakennettu CSS- ja JavaScript-tiedostojen optimointityökalu ne tallentaa. Tällä tavoin pystyin siis kiertämään private-tiedostojärjestelmän aiheuttaman ongelman, joka tekee sisäänrakennetusta CSS:n ja JavaScriptin optimointityökalusta hyödyttömän. Asennetun moduulin avulla sain siis pakattua ja yhdistettyä kaikkialta järjestelmästä tulevat CSS- ja JavaScript-tiedostot, jolloin HTTP-pyyntöjen määrä väheni ja tiedostojen tiedostokoko pieneni. [60.]

Kytkin järjestelmän asetuksista vielä pois päältä lokin kirjoittamisen tietokantaan ja sen näyttämisen sivustolla. Lokiin kirjautuvat tapahtumia ei siis tämän muutoksen jälkeen ladata esiin käyttäjille eikä niiden tallentaminen enää aiheuta kuormaa tietokantaan.

Itse sovelluksen optimoinnin lisäksi täytyi palvelimella oleviin, sovelluksen toimintaan liittyviin tiedostoihin tehdä muutoksia. Palvelimella olevasta Apahce-ohjelmistosta täytyi kytkeä päälle mod_expires-laajennus, jotta staattisiin tiedostoihin voisi asettaa pitkälle tulevaisuuteen voimassa olevat otsikot. Tämä tarkoittaa sitä, että kun tiedostossa on pitkälle tulevaisuuteen ulottuva voimassaoloaika, selain voi tallentaa sen välimuistiinsa mahdollisimman pitkäksi aikaa eikä sen tarvitse joka sivunlatauksella hakea samaa tiedostoa uudestaan. Mod_expires-laajennus mahdollistaa näiden otsikoiden määrittelyn htaccess-tiedostossa. [61.] Nämä htaccess-tiedostot ovat Apachen käyttämiä asetustiedostoja, joiden avulla voidaan määritellä kansiokohtaisia muutoksia järjestelmään, jolloin tehdyt muutokset koskevat vain kyseisessä kansiossa olevaa asennusta. [62.]

Toinen muutos palvelinpuolen tiedostoihin liittyi sivuston tiedostojen Gzip-pakkaukseen. Tämän pakkauksen päälle kytkemiseksi täytyi kytkeä päälle toinen Apachen laajennus, mod_deflate, ja kirjoittaa vaaditut asetusten muutokset htaccess-tiedostoon. [61.]

Kuten taulukosta 1 nähdään, tiedostojen pakkaaminen Gzip-pakkaustekniikalla toi huomattavan säästön tiedostojen koossa. Kaikkien tiedostojen koko, huomioon ottamatta kuvia, laski vähintään yli 60 prosenttia, ja parhaimmillaan saavutettiin yli 80 prosentin säästö. Tämä merkitsee suurilla käyttäjämäärillä jo huomattavaa määrää bittejä, joten palvelin pystyy palvelemaan samalla siirtonopeudella suurempaa määrää käyttäjiä.

Taulukko 1. Lääkärikeskuksen ekstranetverkkopalvelun Gzip-pakkauksen mittaustulokset.

Tiedostokoko ennen Gzip-pakkausta		
Etusivun HTML	48,6 kB	
CSS-tiedostot yhteensä	100,6 kB	
JavaScript-tiedostot yhteensä	63,8 kB	
Tiedostokoko Gzip-pakkauksen jälkeen		
Etusivun HTML	8,7 kB	–82 %
CSS-tiedostot yhteensä	19,1 kB	–81 %
JavaScript-tiedostot yhteensä	24,4 kB	–62 %

YSlow'n sisältämän Yahoo! Smush.it -työkalun avulla optimoin sivuston teeman sisältämät kuvat niiden kuvanlaadun heikentymättä. Työkalu pystyi pienentämään kuvien tiedostokokoa 41,42 %, eli saavutin jälleen huomattavan säästön tiedostokoossa varsinkin suuria käyttäjämääriä silmälläpitäen. [48.]

Tällaiselle sivustolle, jossa käyttäjinä on vain kirjautuneita käyttäjiä, Authcache-moduulisi olisi mitä luultavimmin tuonut lisää nopeutta sisään kirjautuneille. Jätin kuitenkin sen pois projektin tiukan aikataulun ja moduulin asetusten paljon aikaa vievän säätämisen takia. Monimutkaisen asennuksen lisäksi sivustolle tulee vain kaksi eri käyttäjäroolia, kun Authcache taas on suunniteltu sivustoille, joissa on useampia käyttäjärooleja ja näille kaikille moduuli tekee oman välimuistisäiliönsä. Moduulin monimutkaisuuden ja projektin yksinkertaisuuden takia sovelluksen ja sen optimoinnin pitäminen yksinkertaisempana oli parempi vaihtoehto. [56.]

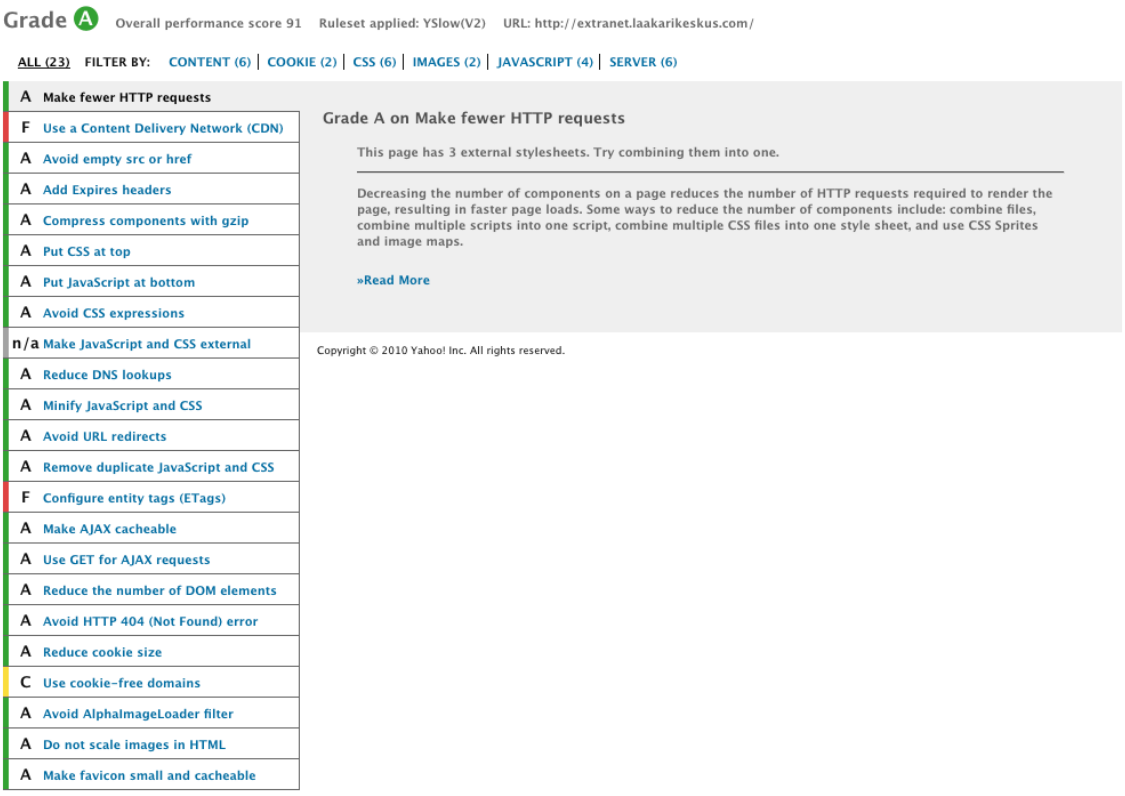
Panels-moduulissa, jota käytetään etusivulla, on oma yksinkertainen välimuistinsa. Jätin ominaisuuden kuitenkin kytkemättä ennen ensimmäistä kuormitustestausta ja suoritin lisätestauksen tämän välimuistin ollessa päällä saadakseni selville, onko sillä suuri vaikutus moduulia käyttävän sivuston toimintaan. Saatujen tulosten valossa voidaan siis todeta, tuoko tämän välimuistin käyttäminen suuria etuja eli kannattaako se huomioida myös tulevissa projekteissa. Asetin Panels-moduulin välimuistin elinajaksi viikon jälkimmäiseen rasitustestaukseen. Panels-moduulin välimuistin päälle kytkemisen lisäk-

si viimeiseen rasitustestaukseen asensin Memcache-moduulin nähdäkseni, tuoko se suorituskykyä lisää vaikka en erityisesti säädä sen asetuksia.

5.5 Optimoidun version mittaustulokset

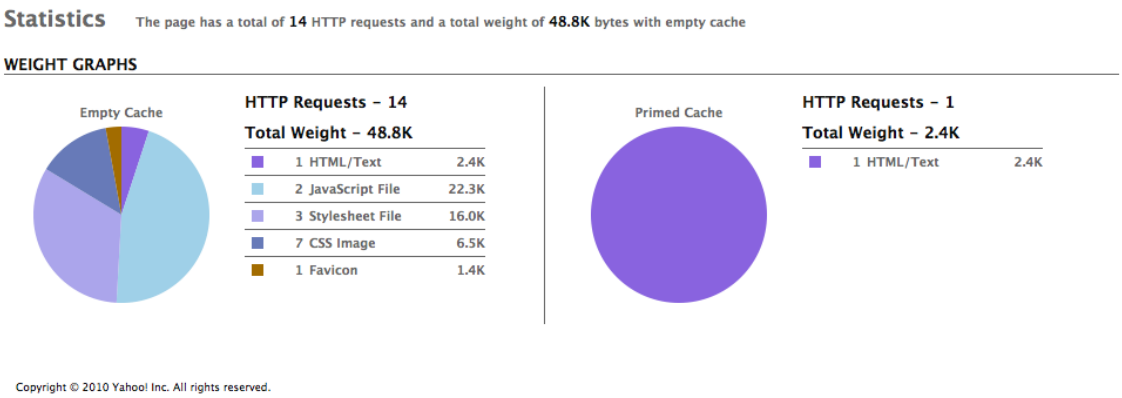
Optimoinnin jälkeen testasin sivuston jälleen Firefox-selaimen YSlow-laajennusosalla. Optimoinnin tulokset olivat selvästi nähtävissä tuloksessa, ja vain säännöt, joiden soveltaminen ei ollut järkevää tähän verkkopalveluun, kuten CDN-järjestelmän käyttäminen, saivat huonoimman mahdollisen arvosanan eli F-kirjamen.

Tuloksista (kuvio 14) nähdään, että yleisarvosanaksi sivusto sai A:n eli parhaan mahdollisen ja yhteispisteiksi tuli 91 pistettä sadasta. Sivuston lataamiseksi tarvittavat HTTP-pyynnöt olivat vähentyneet ja tiedostot pakattiin ennen lähettämistä selaimelle. Tiedostojen otsakkeet oli asetettu pitkälle tulevaisuuteen, jolloin selain pystyi tallentamaan niitä välimuistiinsa ja JavaScript-tiedostot lisättiin vasta sivun lopussa, toisin kuin optimoimattomassa sivussa.



Kuvio 14. YSlow'n tulokset optimoinnin jälkeen.

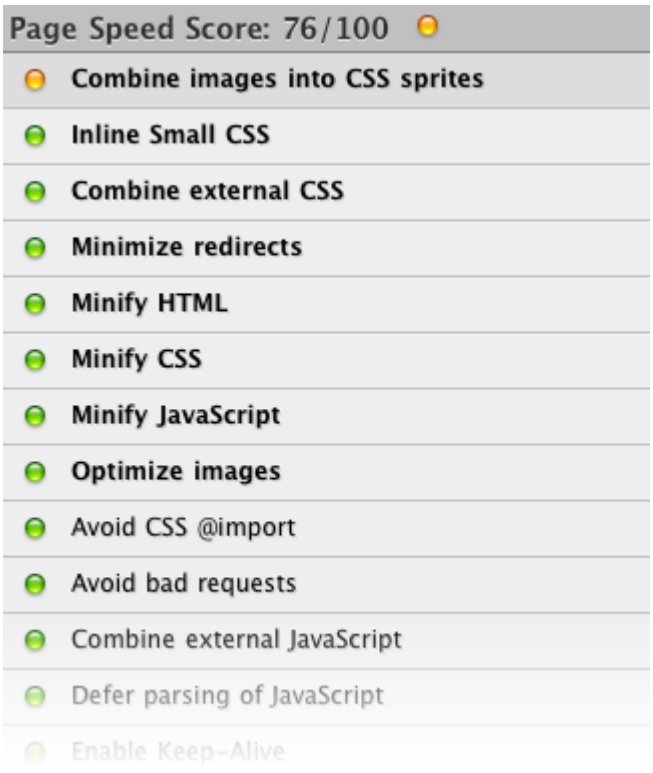
Kuviosta 15 käy vielä paremmin selville, kuinka paljon hyötyä optimoinnista oli sivuston tiedostokokoon pienentämisessä ja HTTP-pyyntöjen määrän vähentämisessä. Optimoimattomaan sivustoon verrattuna sivuston koko oli pienentynyt 350 kilotavusta 48,8 kilotavuun ja HTTP-pyyntöjen määrä vähentynyt 66:sta 14 pyyntöön. Selaimen välimuisti oli myös otettu hyötykäyttöön optimoinnin jälkeen, kuten ympyrädiagrammeista voidaan nähdä. Ensimmäisellä kerralla kun selain lataa sivuston, suoritetaan 14 HTTP-pyyntöä, mutta seuraavalla latauksella tarvitaan enää yksi pyyntö, jolla haetaan sivuston HTML palvelimelta ja kaikki muut tiedostot voidaan tarjota käyttäjälle suoraan selaimen välimuistista.



Kuvio 15. YSlow'n keräämät tiedot sivuston toiminnasta selaimessa optimoinnin jälkeen.

Google Page Speed antoi optimoidulle sivustolle arvosanaksi 76 pistettä sadasta. Tulos oli siis jossain määrin huonompi kuin YSlow'n antama kokonaispistemäärä, mutta silti huomattavasti parempi kuin sivuston arvosana ennen optimointia eli 30/100.

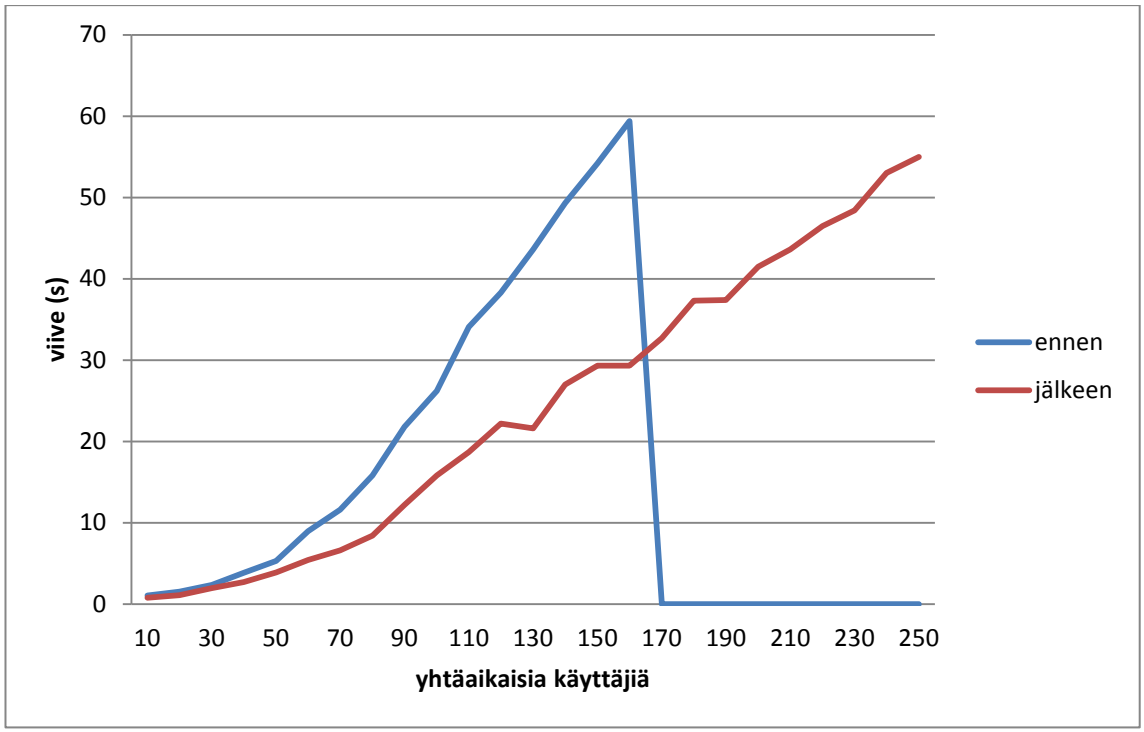
Kuviosta 16 nähdään että Google Page Speed ei antanut yhtään punaista indikaattoria testattujen sääntöjen eteen, mutta kuvien määrä sivustolla oli testiohjelman mielestä vielä liian suuri (seitsemän), ja se antoi tämän säännön noudattamisesta keltaisen indikaattorin. Valitettavasti teemaan käytettyjen kuvien vähentäminen sivustolla yhdistämällä niitä sprite-grafiikaksi ei ollut mahdollista kaikissa kuvissa, joten sivustolle oli jätettävä muutamia yksittäisiä kuvia sprite-tiedoston lisäksi.



Kuvio 16. Google Page Speed -työkalun antamat tulokset optimoinnin jälkeen.

Google Page Speedin jälkeen aloitin sivuston rasitustestauksen. Aluksi suoritin sivustolle Load Impact -testauksen samoilla asetuksilla kuin ennen optimointia. Tällä kertaa palvelin kesti kaatumatta koko testauksen ajan, mikä kertoi siitä, että sivuston optimoinnilla oli positiivinen vaikutus suorituskykyyn.

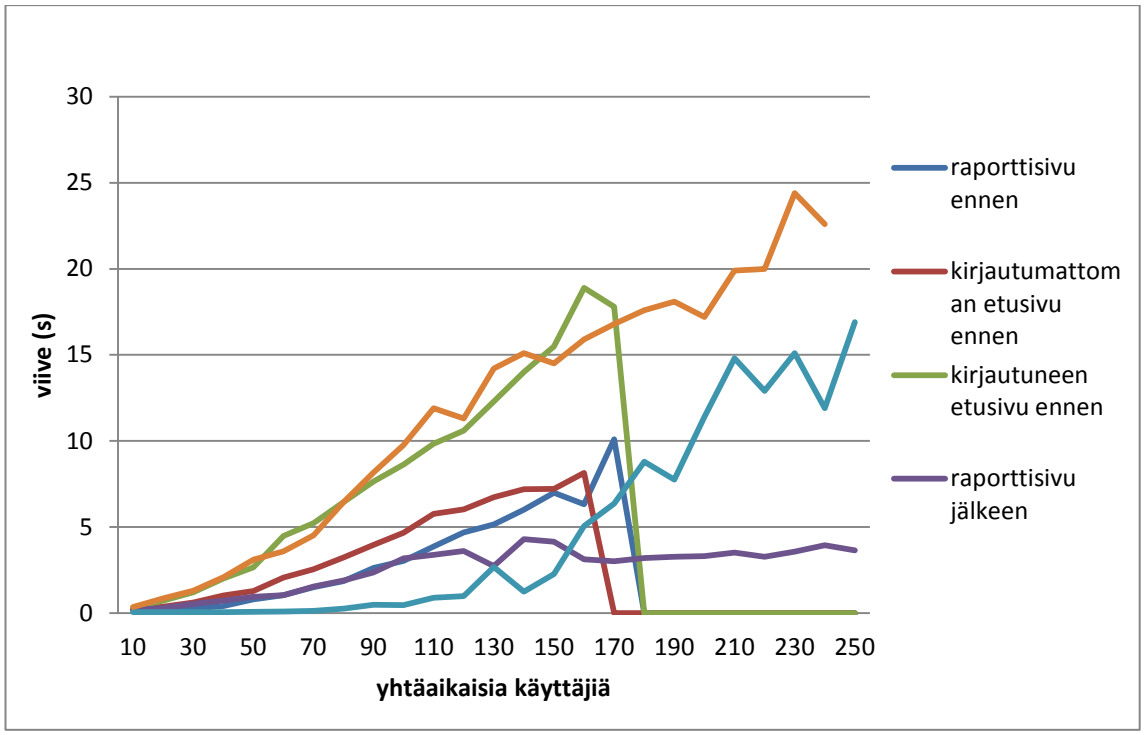
Kun kahden mittauksen tulokset laitetaan samaan kaavioon (kuvio 17), nähdään selvästi, kuinka paljon suorituskykyisempi optimoitu sivusto on. Optimoidun sivuston kuvaaja (punainen käyrä) on selvästi optimoimatonta käyrää (sininen käyrä) loivempi ja pysyy testin loppuun saakka pienemmissä lukemissa kuin optimoimaton. Käyrän muoto on myös suorempi kuin ensimmäisessä testauksessa, mikä kertoo siitä, että sivuston suorituskyky heikkenee lähes lineaarisesti. Tämä on Load Impactin sivuston mukaan hyvä ominaisuus sivuston toimintaa ajatellen ja vaikea ominaisuus toteuttaa teknisesti [59]. Tätä projektia aikaisempien sivustojen mittaustulosten valossa on havaittu, että Pressflow'n ja Varnishin yhdistelmällä palvelin pystyy reagoimaan huomattavasti suurempiin käyttäjämääriin, ilman että palvelin kaatuu. Palvelin ei siis kaadu helposti suurenkaan käyttäjämäärän alla, vaikka viive nousisi esimerkiksi viiden minuutin paikkeille, toisin kuin optimoimattomalle sivustolle kävi jo 160 käyttäjän kohdalla. [56.]



Kuvio 17. Sivuston viiveet ennen optimointia ja sen jälkeen.

Ensimmäisessä testissä määrittelin hyväksyttäväksi viiveeksi sivuston toiminnalle 6–7 sekuntia, ja silloin sain tulokseksi noin 55 yhtäaikaista käyttäjää. Optimoinnin jälkeen sivuston viive on näissä lukemissa noin 70 käyttäjän kohdalla, eli optimoitu sivusto kestää noin 15 yhtäaikaista käyttäjää enemmän kuin optimoimaton. Tulos ei ollut aivan yhtä hyvä kuin olisin toivonut, mutta ymmärrettävä, koska kirjautuneelle käyttäjälle sivuston toiminta ei juuri poikkea optimoimattomasta välimuistin puutteen vuoksi [16, s. 349; 56]. Johtopäätöksenä tästä tuloksesta on se, että sivuston suorituskyvyn parantamiseksi välimuistin kytkeminen päälle sovelluksessa on erityisen tärkeää.

Yksittäisten sivujen viiveet (kuvio 18) antavat selkeämmän kuvan siitä, mihin optimointi on vaikuttanut eniten. Kuviossa 18 ovat samassa kaaviossa ensimmäisen mittauksen tulokset ja optimoinnin jälkeiset käyrät samoilta sivuston osilta. Tästä nähdään hyvin, että kirjautuneen käyttäjän etusivun latausaika on pysynyt lähes entisellään optimoinnista huolimatta, kun taas kirjautumattoman käyttäjän etusivun toiminta nopeutui varsinkin mittauksen alkupäässä. Molemmilla sivuilla on käytössä Panels-moduuli, ja kirjautuneelle käyttäjälle suurin osa sivun elementeistä ei tule välimuistista [16, s. 349; 56].



Kuvio 18. Yksittäisten sivujen viive ennen optimointia ja sen jälkeen.

Raporttien listaussivun latausajat seurasivat aluksi optimoimattoman sivun käyrää, mutta noin sadan yhtäaikaisen käyttäjän kohdalla käyrät erkanivat ja optimoidun raporttisivun kuvaaja pysyi alle viiden sekunnin viiveessä testin loppuun saakka. Raporttien listaussivulla ei ole käytössä Panels-moduulia eikä sisältö tule myöskään välimuistista, koska käyttäjä on kirjautuneena sisään sivua tarkastellessaan [16, s. 349; 58].

Tästä tuloksesta voidaan päätellä, että yksi suurimmista suorituskykyä heikentävistä osista sivustolla on Panels-moduuli. Sen toiminnallisuuden korvaaminen jollakin muulla moduulilla, kuten Views-moduulilla, joka on käytössä raporttien listaussivulla, voisi näiden tulosten valossa vaikuttaa myönteisesti sivuston suorituskykyyn.

Viimeiseksi testasin sivustoa vielä Siegellä samoilla asetuksilla kuin ennen optimointia. Tulokset (liite 2) kertoivat kirjautumattoman käyttäjän etusivun keventyneen huomattavasti optimoinnin ansiosta ja sivuston latausajat olivat huomattavasti pienempiä verrattuna optimoimattomaan versioon. Kymmenellä yhtäaikaisella käyttäjällä sivuston keskimääräinen latausaika oli vain 0,07 sekuntia, kun optimoimattomalla sivulla sama tulos oli 0,75 sekuntia. Kun sivustoa kuormittavien yhtäaikaisten käyttäjien määrä nostettiin 12:een, keskimääräinen latausaika ei noussut 0,07 sekunnista. Ennen optimoin-

tia 12 yhtäaikaista käyttäjää sai sivun latausajaksi 0,97 sekuntia, jonka määrittelin hyväksyttävän latausajan rajaksi. Optimoinnin jälkeen edes 50 yhtäaikaista käyttäjää ei saanut latausaikaa nostettua yhden sekunnin yli, vaan se jäi 0,33 sekuntiin. Suorituskyky oli siis parantunut huomattavasti. Suoritin vielä lisätestin selvittääkseni, kuinka monta yhtäaikaista käyttäjää vaaditaan, että sivun latausaika ylittäisi yhden sekunnin. Sadalla yhtäaikaisella käyttäjällä latausaika ylitti yhden sekunnin 0,03 sekunnilla eli optimoitu kirjautuneen käyttäjän etusivu kesti sata yhtäaikaista käyttäjää latausaikojen karkaamatta liian korkeiksi, kun taas optimoimaton sivu kesti vain 12 yhtäaikaista käyttäjää. Sivuston keskimääräinen yhtäaikaisten yhteyksien määrä pysyi myös hyvin hallinnassa, eikä palvelin siis ollut yhtä suuren rasituksen alla kuin ennen optimointia [59].

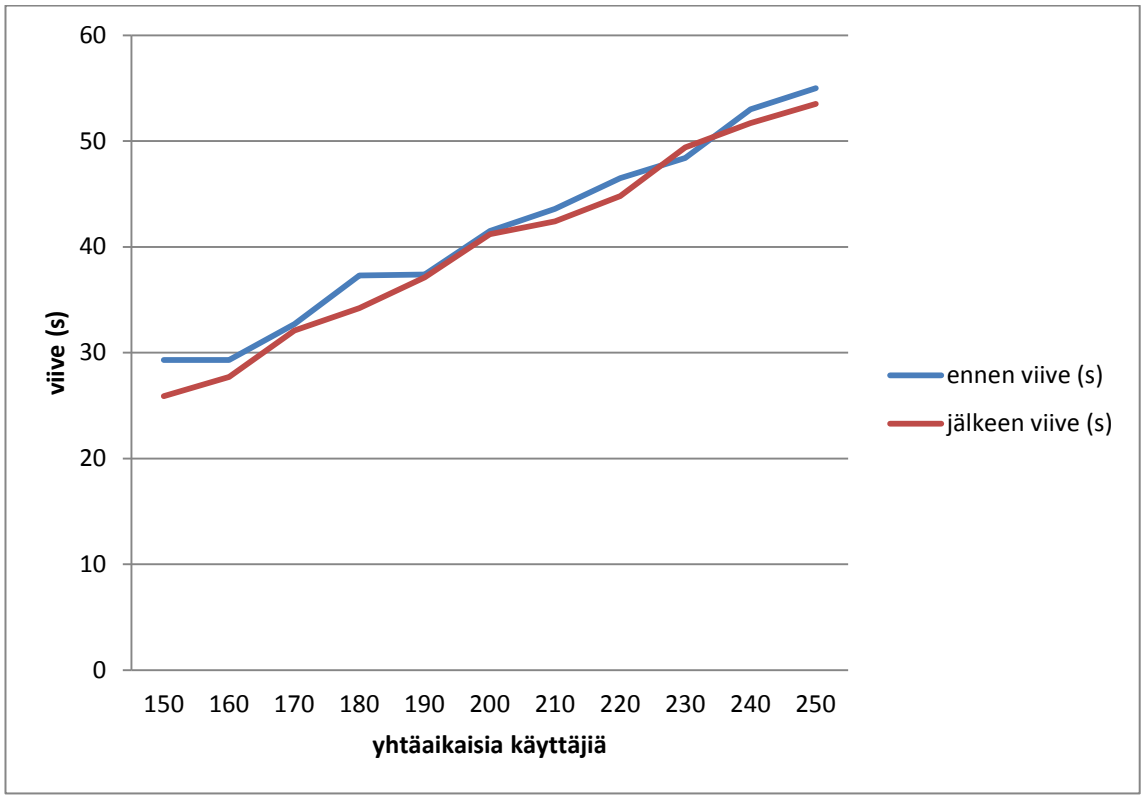
Vertasin vielä Load Impactin antamia tuloksia Siegen antamiin ja havaitsin, että ne täsmäsivät melko hyvin. Kirjautumattoman käyttäjän etusivu ennen optimointia tosin poikkesi hieman, eli Load Impact antoi sivuston keskimääräiseksi latausajaksi yhden sekunnin vasta 40 yhtäaikaisen käyttäjän kohdalla, kun taas Siege sai vastaavan tuloksen jo 12 käyttäjän kohdalla. Saman sivun tulokset optimoinnin jälkeen erosivat vähemmän: Load Impact antoi yhden sekunnin latausajan 120 käyttäjän kohdalla, kun taas Siege antoi saman tuloksen 100 yhtäaikaisen käyttäjän rasitteen alla. Poikkeavuudet voidaan osaksi selittää siellä, että Siege ajettiin lokaalisti, kun taas Load Impact suoritti rasitustestin Tukholmassa sijaitsevalta palvelimelta [56].

5.6 Lisämittaukset

Kuten aiemmin totesin, päätin testata sivuston suorituskykyä pienen lisäoptimoinnin jälkeen sivustolla. Kytkin etusivun Panels-moduulilla tehdystä sivusta yksinkertaisen välimuistin päälle ja asetin sen elinajaksi pisimmän tarjotun ajan. Tämän jälkeen latasin ja asensin sivustolle Memcache-moduulin ja ilman minkäänlaista asetusten muokkausta. Kävin sivustoa läpi, jotta mahdollisimman paljon sen elementeistä tallentuisi välimuistiin ja aloitin uuden rasitustestin sivustolla. Käytin testaukseen Load Impactia, jotta voisin vertailla kirjautuneen ja kirjautumattoman käyttäjän testitulosten eroja.

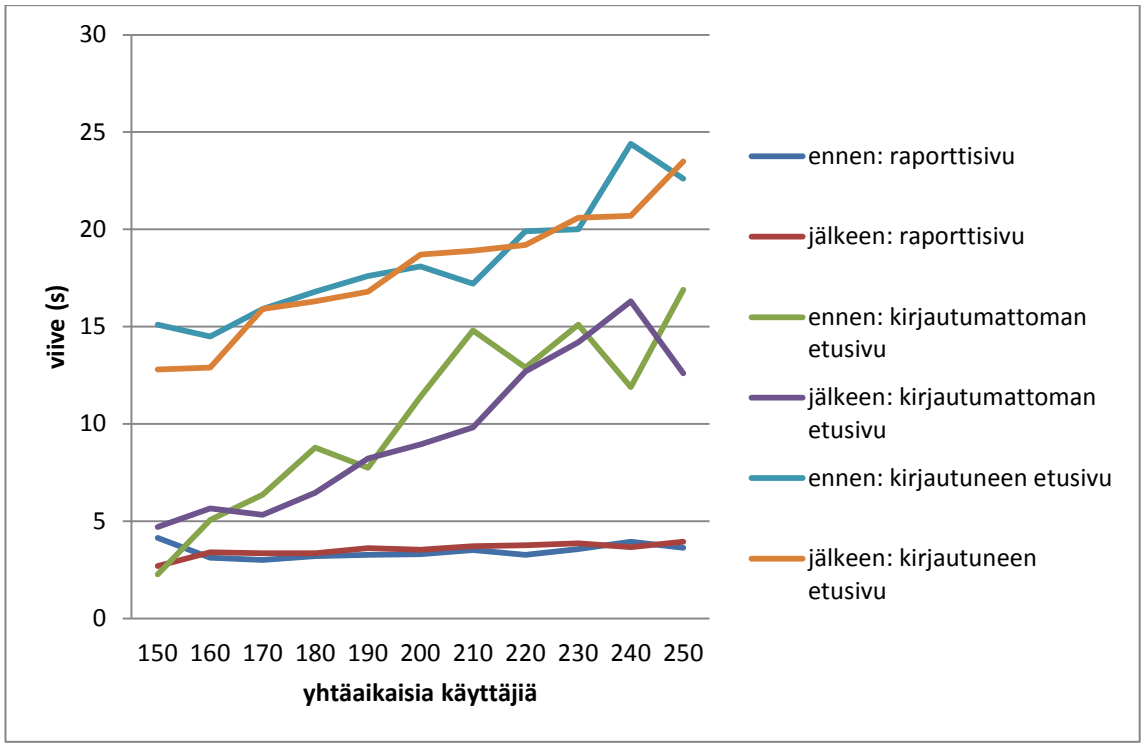
Kuten kuviosta 19 käy selville, kokonaisviiveen määrä ei laskenut käytännössä lainkaan uusien optimointitoimenpiteiden ansiosta. Syytä tähän voi vain arvailla, koska käyrissä

ei näy lähes mitään eroa, mutta todennäköistä on, että Memcache-moduuli ei ainakaan suoraan ilman asetusten muokkaamista parantanut suorituskykyä.



Kuvio 19. Lisäoptimoinnin jälkeiset viiveet sivustolla.

Yksittäisten sivujen viiveet ennen lisäoptimointia ja sen jälkeen (kuvio 20) eivät myöskään kerro merkittävästä parannuksesta suorituskyvyssä miltään osin, vaan käyrät noudattelevat aikaisempia mittaustuloksia pieniä vaihteluita lukuun ottamatta. Etusivuilla käyttöönotetulla Panels-moduulin välimuistilla ei näytä myöskään olevan vaikutusta sivujen viiveisiin. Epäilen, että moduuliin sisäänrakennetun välimuistin vaikutus jäi minimaaliseksi jo käytössä olevien välimuistitasojen takia. Näitä tasoja ovat Drupalin ytimen sisäänrakennettu välimuisti ja käänteisen välityspalvelimen tarjoama välimuistitaso. Tästä tuloksesta tulin siihen johtopäätökseen, että Panels-moduulin sisäänrakennettu välimuisti ei lisää sivuston suorituskykyä kovin paljon, varsinkaan jos sivusto muilta osin hyvin optimoitu ja Drupalin ytimen mukana tuleva välimuisti on kytketty päälle. Toinen tuloksista selvinnyt huomion arvoinen seikka on Memcache-moduulin asetusten tarkempi tarkastelu, kun halutaan saavuttaa suorituskykyparannuksia moduulin avulla sivustolle.



Kuvio 20. Yksittäisten sivujen viiveet lisäoptimoinnin jälkeen.

6 Yhteenveto

Kun verkkosivua optimoidaan suorituskykyisemmäksi, tulee vähentää sovelluksen aiheuttamia HTTP-pyyntöjä palvelimelle yhdistämällä sivustolla olevia ladattavia elementtejä. Ulkoasussa käytettävät kuvat tulisi yhdistää mahdollisuuksien mukaan yhdeksi kuvaksi, ja kaikki sivustolla käytetyt CSS-tiedostot tulisi yhdistää siten, että parhaimmassa tapauksessa selaimen tarvitsee tehdä vain yksi HTTP-pyyntö palvelimelle tyylimääritysten hakemiseksi. Sovelluksesta tulisi poistaa kaikki ylimääräiset ja sovelluksen kehitykseen tarkoitetut, julkisessa sivustossa turhat moduulit, koska niitä ei tarvita sivuston toiminnan kannalta ja ne hidastavat sivuston toimintaa. Tiedostojen koko tulisi optimoida mahdollisimman pieneksi. Tekstitiedostojen kohdalla tulisi käyttää kielten tarjoamia lyhenteitä ja pakata ne käyttämällä Gzip-pakkausta ennen selaimelle lähettämistä. Mediatiedot, kuten kuvat, musiikki ja videot, tulisi pakata mahdollisimman pieneen kokoon.

Sovelluksen käyttämien tiedostojen optimoinnin lisäksi täytyy ottaa huomioon itse sovelluksen toiminta ja sen optimointi. Raskaiden, jatkuvasti toistuvien tietokantahakujen ja hitaasti toimivan koodin suorittaminen hidastaa verkkosivun toimintaa ratkaisevasti. Tämän takia sovellusta optimoitaessa tällaiset toiminnallisuudet täytyy rakentaa uudestaan suorituskykyisemmiksi tai tallentaa niiden tulokset välimuistiin. Välimuistin avulla tulokset voidaan hakea nopeasti pienellä viiveellä palvelimelta eikä palvelimen tarvitse suorittaa näitä hitaita toimenpiteitä jatkuvasti. Välimuistin käyttäminen verkkosivulla on siis suorituskyvyn kannalta erittäin tärkeää, ja parhaassa tapauksessa kaikki sisältö pystytään tarjoamaan käyttäjille välimuistista. Suorituskykyoptimoinnissa tulisi myös ottaa huomioon selaimen välimuisti eli sovelluksen tiedostoille tulisi antaa otsikot, jotka sallivat niiden käytön suoraan selaimen välimuistista tietyn ajan sisällä. Näin selaimen ei tarvitse ladata jokaista tiedostoa uudestaan palvelimelta ensimmäisen latauskerran jälkeen, vaan se voi käyttää välimuistiin tallentamaansa tiedostoa aina, kun sitä kutsutaan sivustolla.

Suorituskykyisen verkkosivun rakentaminen ei ole vaikeaa, mutta siinä harvoin onnistuu, jos unohtaa jokaisen päätöksen kohdalla ajatella toiminnallisuuden vaikutusta nopeuteen. Sivuston tai verkkopalvelun nopeus tulisi pitää mielessä alusta lähtien ja pohtia jokaisen uuden ominaisuuden vaikutusta kokonaisuuteen. Ennen palvelun tai sovelluksen verkkoon päästämistä sen suorituskyky tulisi myös testata, jotta vältyttäisiin

ikäviltä yllätyksiltä ja mahdolliset pullonkaulat suorituskyvyssä löytyisivät, ennen kuin niistä aiheutuu ongelmia asiakkaalle, käyttäjille ja kehittäjille. Sivuston kehittämisen yhtenä lähtökohtana tulisi siis pitää suorituskykyä käytettävyyden, hienon ulkonäön, turvallisuuden ja toimintavarmuuden lisäksi, sillä usein suorituskykyinen sivusto on myös käytettävyydeltään ja toimintavarmuudeltaan parempi kuin optimoimaton sivusto.

Teeman rakentaminen on kokemusteni mukaan yksi tärkeimmistä elementeistä optimointia ajatellen. Huonosti rakennettu teema heikentää suorituskykyä suuresti, ja sitä on jälkikäteen usein vaikea korjata. Ulkoasussa käytettyjen kuvien aiheuttamat ylimääräiset HTTP-pyynnöt hidastavat sivustoa huomattavasti, joten siitä huolimatta että sprite-grafiikan käyttö ulkoasua rakennettaessa aiheuttaa lisää työtä, se maksaa vaivan takaisin sivuston toiminnan sujuvuutena. Tyylien määrittelyyn käytetyt CSS-tiedostot ja dynaamisuuden mahdollistavat JavaScript-tiedostot tulisi myös suunnitella huolellisesti jälkikäteen tapahtuvaa kehitystä ajatellen ja kirjoittaa selkeästi, mutta lyhyesti. Teeman optimointiin löytyy useita valmiita työkaluja, jotka ovat usein jo sisäänrakennettuja sisällönhallintajärjestelmään, mutta nekään eivät auta, jos kehityksen aikana nopeuteen ei ole kiinnitetty lainkaan huomiota.

Nopeat verkkoyhteydet ovat tuoneet mukanaan dynaamisuuden vaatimuksen verkkosivustoille, joten sovelluksia rakennettaessa helposti unohtuu välimuistin tarpeellisuus. Se koetaan helposti dynaamisuutta rajoittavana pahana, mutta todellisuudessa sen tuomat hyödyt suorituskyvylle ovat vertaansa vailla. Opin arvostamaan hyvin rakennettua järjestelmää, jossa on toimiva ja yksinkertainen välimuistin rakenne, koska se on todella vaikea toteuttaa.

Usein yritysmaailmassa sivustot rakennetaan kovalla kiireellä tiukkojen aikataulujen takia eikä aikaa jää testaukselle, mutta tämän työn ohessa opin, kuinka tärkeää testaaminen todellisuudessa on. Drupal-sisällönhallintajärjestelmää käytettäessä Panels-moduulin aiheuttamat suorituskykyä heikentävät ominaisuudet eivät olisi välttämättä selvinneet minulle missään vaiheessa sivustoa rakennettaessa, ellei sen suorituskykyä olisi testattu rastitustestein. Ne paljastivat moduulin heikkoudet verrattuna esimerkiksi Views-moduuliin ja opettivat minulle tärkeän tavoitteen käyttää tätä moduulia sivustoilla mahdollisimman vähän.

Tulevissa projekteissani hyödynnän oppimaani sisällönhallintajärjestelmistä ja niiden optimoinnista, koska olen kokenut hitaan ja huonosti rakennetun sivuston aiheuttamat ongelmat ja tunnistanut monia niistä tätä työtä tehdessäni. Olen myös itse ollut osasyyllinen tiettyjen sivustojen hitauteen, koska olen sortunut huonoihin mutta helppoihin toimintatapoihin verkkosivuja rakentaessani. Tehdessäni käytännön optimointia tätä työtä varten olen kuitenkin oppinut niiden vaarallisuuden ja ymmärrän, että myös verkkosivuihin pätee vanha sananlasku: sen minkä taakseen jättää, edestään löytää.

Lähteet

1 Linden, Greg. 2006. Marissa Mayer at Web 2.0. Verkkodokumentti. <http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html>. 9.11.2006. Luettu 7.3.2011.

2 Soprano Brain Alliance on web-pohjaisten sovellusten kehittämisen asiantuntija ja ratkaisutoimittaja. 2011. Verkkodokumentti. <http://www.brainalliance.com>. Luettu 5.4.2011.

3 Schlossnagle, Theo. 2006. Scalable Internet Architectures. United States of America: Developer's Library.

4 Johnston, Mike. 2010. What is a CMS? Verkkodokumentti. <http://www.cmscritic.com/what-is-a-cms/>. 28.7.2010. Luettu 26.3.2011.

5 Most popular CMS (Content Management System). Verkkodokumentti. Codex-m. <http://www.php-developer.org/most-popular-cms-content-management-system/>. Luettu 26.3.2011.

6 About WordPress. Verkkodokumentti. Wordpress.org. <http://wordpress.org/about/>. Luettu 26.3.2011.

7 Mikä on Joomla!? 2005. Verkkodokumentti. Joomlaportal.fi. <http://www.joomlaportal.fi/content/view/93/39/>. 17.11.2007. Luettu 26.3.2011.

8 About Drupal. Verkkodokumentti. Drupal.org. <http://drupal.org/about>. Luettu 26.3.2010.

9 What the Web's most popular sites are running on. 2007. Verkkodokumentti. Pingdom. <http://royal.pingdom.com/2007/02/26/what-the-webs-most-popular-sites-are-running-on/>. 26.2.2007. Luettu 5.4.2011.

10 TIOBE Programming Community Index for April 2011. 2011. Verkkodokumentti. Tiobe Software. <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>. Luettu 8.4.2011.

11 Usage Stats for April 2007. 2007. Verkkodokumentti. The PHP Group. <http://www.php.net/usage.php>. Luettu 8.4.2011.

12 Hogbin, Emma Jane & Käfer, Konstantin. 2009. Front End Drupal. Crawfordsville: Prentice Hall.

13 King, Andrew B. 2008. Website optimization. United States of America: O'Reilly Media.

14 Souders, Steve. 2007. High Performance Web Sites. United States of America: O'Reilly Media.

15 Introduction. 2003. Verkkodokumentti. Gzip.org. <http://www.gzip.org/>. 27.7.2003. Luettu 26.3.2010.

16 VanDyk, John K. 2008. Pro Drupal Development. United States of America: Apress.

17 Edward Buck. 2009. Is Drupal slow and bloated? Verkkodokumentti. <http://www.nixer.org/is-drupal-slow-bloated>. 3.4.2009. Luettu 23.3.2011.

18 Brief Drupal Optimization Tips – Intermediate and above. 2005. Verkkodokumentti. Drupal.org. <http://drupal.org/node/32091>. 24.9.2005. Luettu 27.3.2011.

19 Less is more: Keeping down the number of modules by extending existing ones. 2009. Verkkodokumentti. 2bits.com, Inc. <http://2bits.com/articles/less-more-keeping-down-number-modules-extending-existing-ones.html>. 19.6.2009. Luettu 1.4.2011.

20 Devel. 2003. Verkkodokumentti. Drupal.org. <http://drupal.org/project/devel>. 28.9.2003. Luettu 26.3.2011.

21 Coles, Malcom. Google Sprite September 2009. 2009. Verkkodokumentti. <http://www.malcolmcoles.co.uk/blog/google-sprite-september-2009/>. 9.2009. Luettu 26.3.2011.

22 JPEG (.jpg) -tiedostot. Verkkodokumentti. Adobe Inc. <http://help.adobe.com/fi_FI/InDesign/5.0/help.html?content=WSa285fff53dea4f8617383751001ea8cb3f-6bce.html>. Luettu 27.3.2011.

23 Siegchrist, Gretchen. 2011. How to Put Video on a Website. Verkkodokumentti. About.com. <http://desktopvideo.about.com/od/videoonyourwebsite/ht/video-on-website.htm>. Luettu 4.4.2011.

24 Cascading Style Sheets. Verkkodokumentti. The World Wide Web Consortium. <http://www.w3.org/Style/CSS/>. 4.3.2011. Luettu 7.3.2011.

25 CSS-opas. 2010. Verkkodokumentti. KK Mediat. <http://www.2kmediat.com/css/syntaksi.asp>. Luettu 9.3.2011.

26 Fusion. 2009. Verkkodokumentti. Drupal.org. <http://drupal.org/project/fusion>. 30.6.2009. Luettu 9.3.2011.

27 Butcher, Matt. 2008. Theming Modules in Drupal 6. Verkkodokumentti. <http://www.packtpub.com/article/theming-modules-in-drupal-6>. 5.2008. Luettu 10.3.2011.

28 Doing CSS aggregation in your theme. 2010. Verkkodokumentti. Drupal.org. <http://drupal.org/node/254780>. 14.2.2010. Luettu 9.3.2011.

29 w3compier. 2009. Verkkodokumentti. Port80 Software Inc. <http://w3compiler.com/specs/>. Luettu 26.3.2011.

30 How do I pass variables between two pages? (GET method). 2006. Verkkodokumentti. <http://www.tek-tips.com/faqs.cfm?fid=5442>. 24.4.2006. Luettu 5.4.2011.

31 Javascript Libraries. 2011. Verkkodokumentti. About.com. <http://javascript.about.com/od/javascriptlibraries/Javascript_Libraries.htm >. Luettu 26.3.2011.

32 CSS Gzip. 2009. Verkkodokumentti. Drupal.org. <http://drupal.org/project/css_gzip>. 24.3.2009. Luettu 10.3.2011.

33 GNU Gzip. 2010. Verkkodokumentti. Free Software Foundation Inc. <http://www.gnu.org/software/gzip/>. 21.1.2010. Luettu 10.3.2011.

34 Introducing JSON. Verkkodokumentti. json.org. <http://json.org/>. Luettu 26.3.2011.

35 Lindgren, Santeri. 2011. Ohjelmistoarkkitehti, Soprano Brain Alliance Oy, Helsinki. Kuva ja keskustelu välimuistien eroista 16.3.2011.

36 Ellison, Jonah. 2009. Authenticated User Page Caching (Authcache). Verkkodokumentti. Drupal.org. <http://drupal.org/project/authcache>. 8.3.2009. Luettu 5.4.2011.

37 Drupal Bootstrap Process. 2010. Verkkodokumentti. ITnitwit.com. <http://www.itnitwit.com/content/drupal-6-bootstrap-process>. 14.7.2010. Luettu 5.4.2011.

38 Bendigen, Arto. 2006. Boost. Verkkodokumentti. Drupal.org. <http://drupal.org/project/boost>. 15.10.2006. Luettu 5.4.2011.

39 About Memcache. 2009. Verkkodokumentti. Dormando. <http://memcached.org/about>. Luettu 5.4.2011.

40 Lund-Chaix, Greg. 2009. Pressflow, Varnish and Caching … oh my!. Verkkodokumentti. <http://blogs.osuosl.org/gchaix/2009/10/12/pressflow-varnish-and-caching/>. 12.10.2009. Luettu 25.3.2011.

41 Pressflow makes Drupal scale. Verkkodokumentti. Four Kitchens. <http://fourkitchens.com/pressflow-makes-drupal-scale>. Luettu 26.3.2011.

42 Joomla 1.5 & Drupal 6.1 Performance Comparison. 2008. Verkkodokumentti. SooperThemes. <http://www.sooperthemes.com/blog/joomla-15-drupal-61-performance-comparison.html>. 18.3.2008. Luettu 23.3.2011.

43 Forward and Reverse Proxies. 2011. Verkkodokumentti. <http://httpd.apache.org/docs/2.0/mod/mod_proxy.html#forwardreverse>. 2011. Luettu 25.3.2011.

44 Varnish Cache. Verkkodokumentti. Varnish Software AB. <http://www.varnish-cache.org/>. Luettu 26.3.2011.

45 Lygstol, Kristian. 2010. A Varnish Crash Course for aspiring sysadmins. Verkkodokumentti. <http://kristianlyng.wordpress.com/2010/07/15/varnish-crash-course-for-sysadmins/>. 15.7.2010. Luettu 26.3.2011.

46 Yahoo! YSlow. 2011. Verkkodokumentti. Yahoo! Inc. <http://developer.yahoo.com/yslow/>. Luettu 8.4.2011.

47 YSlow User Guide. 2011. Verkkodokumentti. Yahoo! Inc. <http://developer.yahoo.com/yslow/help/>. Luettu 8.4.2011.

48 Smush.it. 2011. Verkkodokumentti. Yahoo! Inc. <http://developer.yahoo.com/yslow/smushit/>. Luettu 8.4.2011.

49 Page Speed. Verkkodokumentti. Google. <http://code.google.com/speed/page-speed/>. Luettu 8.4.2011.

50 Using Page Speed for Google Chrome. Verkkodokumentti. Google. <http://code.google.com/speed/page-speed/docs/using_chrome.html>. Luettu 8.4.2011.

51 About Load Impact. 2011. Verkkodokumentti. Load Impact. <http://loadimpact.com/info/about.php>. Luettu 8.4.2011.

52 About Siege. Verkkodokumentti. Joe Dog Software. <http://www.joedog.org/index/siege-home>. Luettu 8.4.2011.

53 Kerner, Sean Michael. 2004. Test your Web server: Lay Siege to it!. Verkkodokumentti. <http://www.builderau.com.au/program/web/soa/Test-your-Web-server-Lay-Siege-to-it-/0,339024632,320283262,00.htm>. 23.3.2004. Luettu 8.4.2011.

54 Arvostetut terveys- ja hyvinvointipalvelut. Verkkodokumentti. Helsingin Lääkärikeskus Oy. <http://www.laakarikeskus.com/laakarikeskus/yrityksestamme>. Luettu 8.4.2011.

55 Initial Setup - File system. Verkkodokumentti. PowerfulCMS. <http://www.powerfulcms.com/cms-drupal-initial-setup>. Luettu 8.4.2011.

56 Lindgren, Santeri. 2011. Ohjelmistoarkkitehti, Soprano Brain Alliance Oy, Helsinki. Keskustelut aikana 8.4.–19.4.2011.

57 Leers, Wim. 2008. Improving Drupal's page loading performance. Verkkodokumentti. <http://wimleers.com/article/improving-drupals-page-loading-performance>. 30.1.2008. Luettu 11.4.2011.

58 Rargnar. 2008. How do I interpret my graph? Is it good or bad?. Verkkodokumentti. Load Impact. <http://loadimpact.com/forum/viewtopic.php?id=87>. 25.9.2008. Luettu 11.4.2011.

59 Performance Statistics. Verkkodokumentti. Joe Dog Software. <http://www.joedog.org/index/siege-manual#a08>. Luettu 11.4.2011.

60 Advanced CSS/JS Aggregation. 2011. Verkkodokumentti. Drupal.org. <http://drupal.org/project/advagg>. 19.2.2011. Luettu 8.4.2011.

61 Why drupal sites are too slow? 2008. Verkkodokumentti. Drupal.org <http://drupal.org/node/287115>. 25.7.2008. Luettu 8.4.2011.

62 .htaccess files. Verkkodokumentti. Apache.org. <http://httpd.apache.org/docs/1.3/howto/htaccess.html>. Luettu 9.4.2011.

Siegen tulokset ennen optimointia

10 concurrent users	
Transactions	232 hits
Availability	100%
Elapsed time	29,93 s
Data transferred	3,10 MB
Response time	0,75 s
Transaction rate	7,75 transactions/s
Throughput	0,10 MB/s
Concurrency	5,81
Successful transactions	232
Failed transactions	0
Longest transaction	1,49
Shortest transaction	0,20

12 concurrent users	
Transactions	240 hits
Availability	100%
Elapsed time	29,69 s
Data transferred	3,21 MB
Response time	0,97 s
Transaction rate	8,08 transactions/s
Throughput	0,11 MB/s
Concurrency	7,87
Successful transactions	240
Failed transactions	0
Longest transaction	1,96
Shortest transaction	0,21

50 concurrent users	
Transactions	249 hits
Availability	100%
Elapsed time	29,05 s
Data transferred	3,33 MB

Response time	4,83 s
Transaction rate	8,57 transactions/s
Throughput	0,11 MB/s
Concurrency	41,38
Successful transactions	249
Failed transactions	0
Longest transaction	7,42
Shortest transaction	0,45

Siegen tulokset optimoinnin jälkeen

10 concurrent users	
Transactions	574 hits
Availability	100%
Elapsed time	29,84 s
Data transferred	1,49 MB
Response time	0,07 s
Transaction rate	19,24 transactions/s
Throughput	0,05 MB/s
Concurrency	1,32
Successful transactions	574
Failed transactions	0
Longest transaction	0,16
Shortest transaction	0,06

12 concurrent users	
Transactions	616 hits
Availability	100%
Elapsed time	29,45 s
Data transferred	1,6 MB
Response time	0,07 s
Transaction rate	20,92 transactions/s
Throughput	0,05 MB/s
Concurrency	1,49
Successful transactions	616
Failed transactions	0
Longest transaction	0,34
Shortest transaction	0,06

50 concurrent users	
Transactions	1752 hits
Availability	100%
Elapsed time	29,23 s
Data transferred	4,55 MB

Response time	0,33 s
Transaction rate	59,94 transactions/s
Throughput	0,16 MB/s
Concurrency	19,73
Successful transactions	1752
Failed transactions	0
Longest transaction	2,58
Shortest transaction	0,06

100 concurrent users	
Transactions	1818 hits
Availability	100%
Elapsed time	29,93 s
Data transferred	4,72 MB
Response time	1,03 s
Transaction rate	60,74 transactions/s
Throughput	0,16 MB/s
Concurrency	62,42
Successful transactions	1818
Failed transactions	0
Longest transaction	6,43
Shortest transaction	0,06